On the Power of the Randomized Iterate*

DRAFT

Iftach Haitner† Danny Harnik ‡ Omer Reingold†

April 9, 2008

Abstract

We consider two of the most fundamental theorems in Cryptography. The first, due to Håstad, Impagliazzo, Levin and Luby (STOC '89, STOC '90, SIAM J. on Computing '99), is that pseudorandom generators can be constructed from any one-way function. The second, due to Yao (FOCS '82), states that the existence of weak one-way functions (i.e. functions on which every efficient algorithm fails to invert with some noticeable probability) implies the existence of full fledged one-way functions. These powerful plausibility results shape our understanding of hardness and randomness in Cryptography. Unfortunately, the reductions given by Håstad et al. and Yao, are not as security preserving as one may desire. The main reason for the security deterioration is the input blow up in both of these constructions. For example, given one-way functions on n bits one obtains by to Håstad et al. pseudorandom generators with seed length $\Omega(n^8)$.

This work revisits a technique that we call the *randomized iterate*, introduced by Goldreich, Krawczyk and Luby (SIAM J. on Computing '93). This technique was used by Goldreich et al. to give a construction of pseudorandom generators from regular one-way functions. We simplify and strengthen this technique in order to obtain a similar reduction where the seed length of the resulting generators is as short as $\mathcal{O}(n \log n)$ rather than $\Omega(n^3)$ achieved by Goldreich et al. Our technique has the potential of implying seed-length $\mathcal{O}(n)$, and the only bottleneck for such a result is the parameters of current generators against space bounded computations. We give a reduction with similar parameters for security amplification of regular one-way functions. This improves upon the reduction of Goldreich, Impagliazzo, Levin, Venkatesan and Zuckerman (FOCS '90) in that the reduction does not need to know the regularity parameter of the functions (in terms of security, the two reductions are incomparable). In addition we use the randomized iterate to show a construction of a pseudorandom generator based on an exponentially hard one-way function that has seed length of only $\mathcal{O}(n^2)$. This improves a recent result of Holenstein (TCC '06) that shows a construction with seed length $\mathcal{O}(n^5)$ based on such one-way functions. Finally, we show that the randomized iterate may even be useful in the general context of Håstad et al. In Particular, we use the randomized iterate to replace the basic building block of the Håstad et al. construction. Interestingly, this modification improves efficiency by an n^3 factor and reduces the seed length to $\mathcal{O}(n^7)$ (which also implies improvement in the security of the construction).

Keywords: cryptography, pseudorandom generator, one-way functions, hardness amplification.

*Preliminary versions of this paper appeared as [HHR06b] and [HHR06a].

†Weizmann Institute of Science, Rehovot, Israel. E-mail: {iftach.haitner,omer.reingold}@weizmann.ac.il. Research supported by grant no. 1300/05 from the Israel Science Foundation.

‡IBM Haifa Research Labs. danny.harnik@gmail.com. Research conducted while at the Weizmann Institute and the Technion.

Contents

1 Introduction

In this work we address two fundamental problems in cryptography, constructing pseudorandom generators from one-way functions and transforming weak one-way functions into strong one-way functions. The common thread linking the two problems in our discussion is the technique we use. This technique that we call the *randomized iterate* was introduced by Goldreich, Krawczyk and Luby [GKL93] in the context of constructing pseudorandom generators from regular one-way functions. We revisit this method, both simplify existing proofs and utilize our new view to achieve significantly better parameters for security and efficiency. We further expand the application of the randomized iterate to constructing pseudorandom generators from any one-way function. Specifically we revisit the seminal paper of Håstad, Impagliazzo, Levin and Luby [HILL99] and show that the randomized iterate can help improve the parameters within. We also give significant improvements to the construction of pseudorandom generators from one-way function that have exponential hardness. Finally, we use the randomized iterate method to both simplify and strengthen previous results regarding efficient hardness amplification of regular one-way functions. We start by introducing the randomized iterate in the context of pseudorandom generators, and postpone the discussion on amplifying weak to strong one-way function to Section 1.3.

1.1 Pseudorandom Generators and the Randomized Iterate

Pseudorandom Generators, a notion first introduced by Blum and Micali [BM82] and stated in its current, equivalent form by Yao [Yao82], are one of the cornerstones of cryptography. Informally, a pseudorandom generator is a polynomial-time computable function G that stretches a short random string x into a long string $G(x)$ that "looks" random to any efficient (i.e., polynomial-time) algorithm. Hence, there is no efficient algorithm that can distinguish between $G(x)$ and a truly random string of length $|G(x)|$ with more than a negligible probability. Originally introduced in order to convert a small amount of randomness into a much larger number of effectively random bits, pseudorandom generators have since proved to be valuable components for various cryptographic applications, such as bit commitments [Nao91], pseudorandom functions [GGM86] and pseudorandom permutations [LR88], to name a few.

The first construction of a pseudorandom generator was given in [BM82] based on a particular one-way function and was later generalized in [Yao82] into a construction of a pseudorandom generator based on any one-way permutation. We refer to the resulting construction as the BMY construction. The BMY generator works by iteratively applying the one-way permutation on its own output. More precisely, for a given function f and input x define the i'th iterate recursively as $f^i(x) = f(f^{i-1}(x))$ where $f^0(x) = f(x)$. To complete the construction, one needs to take a hardcore-bit at each iteration. If we denote by $b(x)$ the hardcore-bit of x (take for instance the Goldreich-Levin [GL89] predicate), then the BMY generator on seed x outputs the hardcore-bits $b(f^0(x)), \ldots, b(f^\ell(x))$.

The natural question arising from the BMY generator was whether one-way permutations are actually necessary for pseudorandom generators or can one do with a more relaxed notion. Specifically, is any one-way function sufficient for pseudorandom generators? Levin [Lev87] observed that the BMY construction works for any *one-way function on its iterates*, that is, a one-way function that remains one-way when applied sequentially on its own outputs. A general one-way function, however, does not have this property since the output of f may have very little randomness in it, and a second application of f may be easy to invert. A partial solution was suggested by Goldreich

et al. [GKL93] that showed a construction of a pseudorandom generator based on any regular one-way function (referred to as the GKL generator). A regular function is a function such that every element in its image has the same number of preimages. The GKL generator uses the technique at the core of this work, that we call the ***randomized iterate***. Rather than simple iterations, an extra randomization step is added between every two applications of f. More precisely,

DEFINITION 1.1 (the randomized iterate (informal))
For function f, input x and random vector of hash functions $\overline{h} = (h_1, \ldots, h_\ell)$, recursively define the i'th randomized iterate (for $i \in [\ell]$) by:

$$f^i(x, \overline{h}) = f(h_i(f^{i-1}(x, \overline{h}))) \ ,$$

where $f^0(x, \overline{h}) = f(x)$.

The rational is that $h_i(f^{i-1}(x, \overline{h}))$ is now uniformly distributed, and the challenge is to show that f, when applied to $h_i(f^{i-1}(x, \overline{h}))$, is hard to invert even when the randomizing hash functions $\overline{h} = (h_1, \ldots, h_\ell)$ are made public. Once this is shown, the generator is similar in nature to the BMY generator (the generator outputs $b(f^0(x, \overline{h})), \ldots, b(f^\ell(x, \overline{h})), \overline{h}$).

Finally, Håstad et al. [HILL99] (combining [ILL89, Hås90]), culminated this line of research by showing a construction of a pseudorandom generator using any one-way function (called here the HILL generator). This result is one of the most fundamental and influential theorems in cryptography. It introduced many new ideas that have since proved useful in other contexts, such as the notion of pseudoentropy, and the implicit use of family of pairwise-independent hash functions as randomness extractors. We note that HILL departs from GKL in its techniques, taking a significantly different approach.

1.1.1 The Complexity and Security of the Previous Constructions

While the HILL generator fully answers the question of the plausibility of a generator based on any one-way function, the construction is highly involved and very inefficient. Other than the evident contrast between the simplicity and elegance of the BMY generator to the complex construction and proof of the HILL generator, the parameters achieved in the construction are far worse, rendering the construction impractical.

In practice, it is not necessarily sufficient that a reduction translates polynomial security into polynomial security. In order for reductions to be of any practical use, the concrete overhead introduced by the reduction comes into play. There are various factors involved in determining the security of a reduction, and in Section 2.10 we elaborate on the security of cryptographic reductions and the classification of reductions in terms of their security. In this discussion, however, we focus only on one central parameter, which is the length m of the generator's seed compared to the length n of the input to the underlying one-way function. The BMY generator takes a seed of length $m = \mathcal{O}(n)$, the GKL generator takes a seed of length $m = \Omega(n^3)$ while the HILL construction produces a generator with seed length on the order of $m = \Omega(n^8)$.[1]

The length of the seed is of great importance to the security of the resulting generator. While it is not the only parameter, it serves as a lower bound to how good the security may be. For

[1] The seed length actually proved in [HILL99] is $\mathcal{O}(n^{10})$, however it is mentioned that a more careful analysis can get to $\mathcal{O}(n^8)$. A formal proof for the $\mathcal{O}(n^8)$ seed length construction is given by Holenstein [Hol06a].

instance, the HILL generator on m bits has security that is at best comparable to the security of the underlying one-way function, but on only $\mathcal{O}(\sqrt[8]{m})$ bits. To illustrate the implications of this deterioration in security, consider the following example: Suppose that we only trust a one-way function when applied to inputs of at least 100 bits, then the GKL generator can only be trusted when applied to a seed of length of at least one million bits, while the HILL generator can only be trusted on seed lengths of 10^{16} and up (both being highly impractical). Thus, trying to improve the seed length towards a linear one (as it is in the BMY generator) is of great importance in making these constructions practical.

1.1.2 Exponentially hard one-way functions and improving the seed length

The BMY and GKL generators demonstrate that assuming restrictions on the underlying one-way function allows for great improvement of the seed length (or input blowup). The common theme in these restrictions is that they deal with the *structure* of the one-way function. A different approach was recently taken by Holenstein [Hol06a], that builds a pseudorandom generator from any one-way function with *exponential hardness*, i.e., for some constant C, no algorithm of running-time at most 2^{Cn} inverts the function with probability better than 2^{-Cn}. This approach is different as it discusses *raw hardness* as opposed to structure. The result in [Hol06a] is essentially a generalization of the HILL generator that also takes into account the parameter stating the hardness of the one-way function. In its extreme case where the hardness is exponential, then the pseudorandom generator takes a seed length of $\mathcal{O}(n^5)$. Alternatively, the seed length can be reduced to as low as $\mathcal{O}(n^4 \log^2 n)$ when the resulting generator is only required to have super-polynomial security (i.e. security of $n^{\log n}$). In its other extreme based on a general one-way function (with superpolynomial hardness), [Hol06a] forms a formal proof of the best known seed length for the HILL construction (seed length $\mathcal{O}(n^8)$).

1.2 Our Results on Pseudorandom Generators

Our improvements to the seed length of pseudorandom generators under the various assumptions are summarized in Figure 1. In the upcoming section we elaborate on each of these constructions and highlight the source of the improvements.

Paper	One-way function type	Seed length
[BM82, Yao82]	One-way permutation	$n + o(n)$
[GKL93]	Regular one-way function	$O(n^3)$
This work	Regular one-way function	$O(n \log n)$
[Hol06a]	One-way function with exponential hardness	$O(n^5)$
This work	One-way function with exponential hardness	$O(n^2)$
This work	Regular one-way function with exponential hardness	$O(n)$
[HILL99]	Any one-way function	$O(n^8)$
This work	Any one-way function	$O(n^7)$

Figure 1: Summary of results.

1.2.1 Regular one-way functions

We give a construction of a pseudorandom generator from any regular one-way function with seed length $\mathcal{O}(n \log n)$. We note that our approach has the potential of reaching a construction with a linear seed, the bottleneck being the efficiency of the current bounded-space generators. Our construction follows the randomized iterate method and is achieved in two steps:

- We give a significantly simpler proof that the GKL generator works, allowing the use of a family of hash functions which is pairwise-independent rather than n-wise independent (as used in [GKL93]). This gives a construction with seed length $m = \mathcal{O}(n^2)$ (see Theorem 3.11).
- The new proof allows for the derandomization of the choice of the randomizing hash functions via the *bounded-space generator* of Nisan [Nis92], further reducing the seed length to $m = \mathcal{O}(n \log n)$ (see Theorem 3.12).

The proof method. Following is a high-level description of our proof method. For simplicity we focus on a single randomized iteration, that is on $f^1(x,h) = f(h(f(x)))$. In general, the main task at hand is to show that it is hard to find $f^0(x,h) = f(x)$ when given $f^1(x,h)$ and h. This follows by showing that any procedure A for finding $f(x)$ given $(f^1(x,h),h)$ enables to invert the one-way function f. Specifically, we show that for a random image $z = f(x)$, if we choose a random and independent hash h' and feed the pair (z,h') to A, then A is likely to return a value $f(x')$ such that $h'(f(x')) \in f^{-1}(z)$ (and thus we obtain an inverse of z).

Ultimately, we assume that A succeeds on the distribution of $(f^1(x,h),h)$, and want to prove A is also successful on the distribution of $(f^1(x,h),h')$ where h' is chosen independently of (x,h). Our proof is inspired by a technique used by Rackoff in his proof of the Leftover Hash Lemma (in [IZ89]). Rackoff proves that a distribution is close to uniform by showing that it has *collision-probability* that is very close to that of the uniform distribution.[2] We would like to follow this scheme and consider the collision-probability of the two aforementioned distributions. In our case, however, the two distributions could actually be very far from each other. Yet, with the analysis of the collision-probabilities we manage to prove that the probability of any event under the first distribution is *polynomially related* to the probability of the same event under the second distribution. This proof generalizes nicely also to the case of many iterations.

The derandomization using bounded-space follows directly from the new proof. In particular, consider the procedure that takes two random inputs x_0 and x_1 and random $\overline{h} = (h_1, \ldots, h_\ell)$, and compares $f^\ell(x_0,\overline{h})$ and $f^\ell(x_1,\overline{h})$. This procedure can be run in bounded-space since it simply needs to store the two intermediate iterates at each point. Also, this procedure accepts with probability that is exactly the collision-probability of $(f^\ell(x,\overline{h}),\overline{h})$. Thus, replacing $\overline{h}$ with the output of a bounded-space generator cannot change the acceptance rate by much, and the collision-probability is thus unaffected. The proof of security of the derandomized pseudorandom generator now follows as in the proof when using independent randomizing hash functions.

1.2.2 Exponentially hard one-way functions

We give a construction of a pseudorandom generator from any exponentially hard one-way function with seed length $\mathcal{O}(n^2)$. If the resulting generator is allowed to have only super-polynomial

[2] The collision-probability of a distribution is the probability of getting the same element twice when taking two independent samples from the distribution.

security then the construction gives seed length of only $\mathcal{O}(n \log^2 n)$. Unlike Holenstein's result, our constructions is specialized for one-way functions with exponential hardness. If the function is $2^{\phi(n)}$-one-way, then the result holds only when $\phi(n) \in \Omega(\frac{n}{\log n})$, and does not generalize for use of weaker one-way functions.

The core technique of our construction is once again the randomized iterate. Unfortunately, the last randomized iteration of a general one-way function is not necessarily hard to invert. It may in fact be easy on a large fraction of the inputs. Yet, we manage to prove the following statement regarding the k'th randomized iteration (Lemma 4.1): There exists a set S^k of inputs to f^k such that the k'th randomized iteration is hard to invert over inputs taken from this set. Moreover, the density of S^k is at least $\frac{1}{k}$ of the inputs. So taking a hard core bit of the k'th randomized iteration is beneficial in the sense that this bit will look random (to a computationally bounded observer), just that this will happen only $\frac{1}{k}$ of the time. We first describe a failed attempt to use this weak hardness of inverting the last randomized iterate for a construction based on any one-way function. We then show how to make this approach work assuming that the one-way function has exponential hardness.

In a nutshell, the construction runs m independent copies of the randomized iterate (on m independent inputs). From each of the m copies a hardcore bit is taken from the k'th iteration. This forms a string of m bits, of which $\frac{m}{k}$ are expected to be random looking. In order to get truly pseudorandom bits, one runs a *randomness extractor* on this string and can generate, say, $\frac{m}{2k}$ pseudorandom bits (see below on randomness extractors and pseudorandomness). However, this approach fails since the total number of pseudorandom bits generated (which is the sum of a harmonic progression) is too low, and in particular insufficient to compensate for the mn bits invested in the random seed.

This problem can be remedied by taking more hardcore bits at each iteration. Specifically, if the one-way function has exponential hardness then a linear number of hardcore bits may be taken at each iteration (as was shown in the original paper of Goldreich and Levin [GL89]). Thus taking $m = n$ independent copies, the total number of pseudorandom bits generated can be larger than the seed length. In the overall construction each independent copy runs a constant number of iterations and the seed length of the construction is $O(n^2)$.

On randomness extractors and pseudorandomness The use of randomness extractors in a computational setting, was initiated in [HILL99]. We give a general "uniform extraction lemma" (Lemma 2.7) for this purpose that is proved using a uniform hardcore Lemma of Holenstein from [Hol05]. Note that a similar proof was given independently in [Hol06a].

1.2.3 Any one-way function

The HILL generator takes a totally different path than the GKL generator. We ask whether the technique of randomized-iterations can be helpful for the case of any one-way function, and give a positive answer to this question. Interestingly, this method also improves the efficiency by an n^3 factor and reduces the seed length by a factor of n (which also implies improvement in the security of the construction) over the original HILL generator. All in all, we present a pseudorandom generator from *any* one-way function with seed length $\mathcal{O}(n^7)$ (Theorem 5.10) which is the best known to date.

As mentioned in the case of exponentially hard one-way functions, the hardness of inverting the randomized iterate deteriorates very quickly when using any one-way function. Therefore we use

only the first randomized iterate of a function, that is $f^1(x,h) = f(h(f(x)))$. Denote the degeneracy of y by $\mathrm{Deg}_f(y) = \lceil \log |f^{-1}(y)| \rceil$ (this is a measure that divides the images of f to n categories according to their preimage size). Let b denote a hardcore-bit (again we take the Goldreich-Levin hardcore-bit [GL89]). Loosely speaking, we consider the bit $b(f(x))$ when given the value $(f^1(x,h),h)$ and make the following observation. When $\mathrm{Deg}_f(f(x)) \geq \mathrm{Deg}_f(f^1(x,h))$ then $b(f(x))$ is (almost) fully determined by $(f^1(x,h),h)$, as opposed to when $\mathrm{Deg}_f(f(x)) < \mathrm{Deg}_f(f^1(x,h))$ where $b(f(x))$ is essentially uniform. But in addition, when $\mathrm{Deg}_f(f(x)) = \mathrm{Deg}_f(f^1(x,h))$ then $b(f(x))$ is computationally-indistinguishable from uniform (that is, looks uniform to any efficient observer), even though it is actually fully determined. The latter stems from the fact that when $\mathrm{Deg}_f(f(x)) = \mathrm{Deg}_f(f^1(x,h))$ the behavior is close to that of a regular function.

As a corollary we get that the bit $b(f(x))$ has entropy of no more than $\frac{1}{2}$ (the probability of $\mathrm{Deg}_f(f(x)) < \mathrm{Deg}_f(f^1(x,h))$), but has entropy of at least $\frac{1}{2} + \frac{1}{\mathcal{O}(n)}$ in the eyes of any computationally-bounded observer (the probability of $\mathrm{Deg}_f(f(x)) \leq \mathrm{Deg}_f(f^1(x,h))$). In other words, $b(f(x))$ has entropy $\frac{1}{2}$ but *pseudo-entropy* of $\frac{1}{2} + \frac{1}{\mathcal{O}(n)}$. It is this gap of $\frac{1}{\mathcal{O}(n)}$ between the entropy and pseudo-entropy that eventually allows the construction of a pseudorandom generator.

Indeed, a function with similar properties lies at the basis of the HILL construction. HILL give a different construction that has entropy p but pseudo-entropy of at least $p + \frac{1}{\mathcal{O}(n)}$. However, in the HILL construction the entropy threshold p is unknown (i.e., not efficiently computable), while with the randomized iterate the threshold is $\frac{1}{2}$. This is a real advantage since knowledge of this threshold is essential for the overall construction. To overcome this, the HILL generator enumerates all values for p (up to an accuracy of $\Omega(\frac{1}{n})$), runs the generator with every one of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor n to the seed length as well an additional factor of n^3 to the number of calls to the underlying function f.

On pseudorandomness in NC^1. For the most part, the HILL construction is "depth" preserving. In particular, given two "non-uniform" hints of $\log n$ bits each (that specify two different properties of the one-way function), the reduction gives generators in NC^1 from any one-way function in NC^1. Unfortunately, without these hints, the depth of the construction is polynomial (rather than logarithmic). Our construction eliminates the need for one of these hints, and thus can be viewed as a step towards achieving generators in NC^1 from any one-way function in NC^1 (see [AIK06] for the significance of such a construction).

1.3 One-way Functions - Amplification from Weak to Strong

The existence of one-way functions is essential to almost any task in cryptography (see for example [IL89]) and also sufficient for numerous cryptographic primitives, such as the pseudorandom generators discussed above. In general, for constructions based on one-way functions we use what are called *strong* one-way functions. That is, functions that can only be inverted efficiently with negligible success probability. A more relaxed definition is that of an δ-weak one-way function where $\delta(n)$ is a polynomial fraction. This is a function that any efficient algorithm fails to invert on almost an $\delta(n)$ fraction of the inputs. This definition is significantly weaker, however, Yao [Yao82] showed how to convert any weak one-way function into a strong one. The new strong one-way function simply consists of many independent copies of the weak function concatenated to each other. The solution of Yao, however, incurs a blow-up factor of at least $\omega(1)/\delta(n)$ to the input

length of the strong function[3], which translates to a significant loss in the security (as in the case of pseudorandom generators).

With this security loss in mind, several works have tried to present an efficient method of amplification from weak to strong. Goldreich et al. [GIL+90] give a solution for one-way permutations that has just a linear blowup in the length of the input. This solution generalizes to *known-regular* one-way functions (regular functions whose image size is efficiently computable), where its input length varies according to the required security. The input length is linear when security is at most $2^{\Omega(\sqrt{n})}$, but deteriorates up to $\mathcal{O}(n^2)$ when the required security is higher (e.g., security $2^{\mathcal{O}(n)}$).[4] Their construction uses a variant of randomized iterates where the randomization is via one random step on an expander graph.

1.3.1 Our Contribution to Hardness Amplification

We present an alternative efficient hardness amplification for regular one-way functions. Specifically, in Theorem 6.1 we show that the m'th randomized iterate of a weak one-way function along with the randomizing hash functions form a strong one-way function (for the right parameter m). Moreover, this holds also for the derandomized version of the randomized iterate (Theorem 6.8), giving an almost linear construction. Our construction is arguably simpler and has the following advantages:

1. While the [GIL+90] construction works only for *known* regular weak one-way functions, our amplification works for any regular weak one-way functions (whether its image size is efficiently computable or not).

2. The input length of the resulting strong one-way function is $\mathcal{O}(n \log n)$ regardless of the required security. Thus, for some range of the parameters our solution is better than that of [GIL+90] (although it is worse than [GIL+90] for other ranges).

Note that our method may yield an $\mathcal{O}(n)$ input construction if bounded-space generators with better parameters become available.

The Idea. At the basis of all hardness amplification lies the fact that for any inverting algorithm, a weak one-way function has a set that the algorithm fails upon, called here the ***failing-set*** of this algorithm. The idea is that a large enough number of randomly chosen inputs are bound to hit every such failing-set and thus fail every algorithm. Taking independent random samples works well, but when trying to generate the inputs to f sequentially this rationale fails. The reason is that sequential applications of f are not likely to give random output, and hence are not guaranteed to hit a failing-set. Instead, the natural solution is to use randomized iterations. It might be easy, however, for an inverter to find some choice of randomizing hash functions so that all the iterates are outside of the required failing-set. To overcome this, the randomizing hash functions are also added to the output, and thus the inverter is required to find an inverse that includes the original randomizing hash functions. In the case of permutations it is obvious that outputting the randomizing hash functions is harmless, and thus the k'th randomized iterate of a weak one-way

[3] The $\omega(1)$ factor stands for the logarithm of the required security. For example, if the security is $2^{\mathcal{O}(n)}$ then this factor is of order n.

[4] Loosely speaking, one can think of the security as the probability of finding an inverse to a random image $f(x)$ simply by choosing a random element in the domain.

permutation is a strong one-way permutation. The case of regular functions, however, requires our analysis that shows that the randomized iterate of a regular one-way function remains hard to invert when the randomizing hash functions are public. We also note that the proof for regular functions has another subtlety. For permutations the randomized iterate remains a permutation and therefore has only a single inverse. Regular functions, on the other hand, can have many inverses. This comes into play in the proof, when an inverting algorithm might not return the right inverse that is actually needed by the proof.

A major problem with the randomized iterate approach is that choosing fully independent randomizing hash functions requires an input as long as that of Yao's solution (an input of length $\mathcal{O}(n \cdot \omega(1)/\delta(n))$). What makes this approach appealing after all, is the derandomization of the hash functions using space-bounded generators, which reduces the input length to only $\mathcal{O}(n \log n)$. Note that in this application of the derandomization, it is required that the bounded-space generator not only approximate the collision-probability well, but also maintain the high probability of hitting any failing-set.

We note that there have been several attempts to formulate such a construction, using all of the tools mentioned above. Goldreich et al. [GIL+90] did actually consider following the GKL methodology, but chose a different (though related) approach. Phillips [Phi93] gives a solution with input length $\mathcal{O}(n \log n)$ using bounded-space generators but only for the simple case of permutations (where [GIL+90] has better parameters). Di Crescenzo and Impagliazzo [DI99] give a solution for regular functions, but only in a model where public randomness is available (in the mold of [HL92]). Their solution is based on pairwise-independent hash functions that serve as the public randomness. We are able to combine all of these ingredients into one general result, perhaps due to our simplified proof.

1.3.2 Additional Issues

On non-length-preserving functions. This work focuses on length-preserving one-way functions. We also demonstrate how our proofs may be generalized to use non-length preserving functions. This generalization requires the use of a construction of a family of *almost* pairwise-independent hash functions.

The results in the public randomness model. Similarly to previous works, our results also give linear reductions in the public randomness model. This model (introduced by Herzberg and Luby [HL92]) allows the use of public random coins that are not regarded a part of the input. Our results, however, introduce significant savings in the amount of public randomness that is necessary.

Paper organization. Section 2 includes the formal definitions and notations used throughout this work. In Section 3 we present our construction of pseudorandom generators from regular one-way functions. Section 4 presents the construction based on exponentially hard one-way functions and in particular proves a lemma regarding the hardness of inverting the randomized iterate of a general one-way function (Lemma 4.1). In Section 5 we present our improvement to the HILL construction of pseudorandom generators from any one-way function. Finally, in Section 6 we present our hardness amplification of regular one-way functions.

2 Preliminaries

2.1 Notations

Given two strings x, y of length n, we denote by $\langle x, y \rangle_2$ their inner product modulus two. A function $\mu : \mathbb{N} \to [0,1]$ is *negligible* if for every polynomial p we have that $\mu(n) < 1/p(n)$ for large enough n. Given a function $f : \{0,1\}^* \mapsto \{0,1\}^*$, we denote by $Dom(f)$ and $Im(f)$ the domain and image of f respectively. Let $y \in Im(f)$, we denote the preimages of y under f by $f^{-1}(y)$. The ***d***egeneracy of f on y is defined by $\mathrm{Deg}_f(y) \stackrel{\text{def}}{=} \lceil \log \left| f^{-1}(y) \right| \rceil$. We denote by $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, the ensemble of functions $\left\{ f_n : \{0,1\}^n \to \{0,1\}^{\ell(n)} \right\}_{n \in \mathbb{N}}$.

2.2 Distributions and Entropy

If X is a random variable taking values in a finite set $\mathcal{U}$, then we write $x \leftarrow X$ to indicate that x is selected according to X. If S is a subset of $\mathcal{U}$, then $x \leftarrow S$ means that x is selected according to the uniform distribution on S. We adopt the convention that when the same random variable occurs several times in an expression, all occurrences refer to a single sample. For example, $\Pr[f(X) = X]$ is defined to be the probability that when $x \leftarrow X$, we have $f(x) = x$. We write U_n to denote the random variable distributed uniformly over $\{0,1\}^n$. Given a function $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, we denote by $f(U_n)$ the distribution over $\{0,1\}^{\ell(n)}$ induced by f operating on the uniform distribution. Let D be a distribution over some finite domain X, we use the following measures of entropy:

- The Shannon-entropy of D is $\mathrm{H}(D) = \sum_{x \in X} D(x) \log \frac{1}{D(x)}$.
- The collision-probability of D is $\mathrm{CP}(D) = \sum_{x \in X} D(x)^2$.
- The min-entropy of D is $\mathrm{H}_\infty(D) = min_{x \in X} \log \frac{1}{D(x)}$.

Two distributions P and Q over Ω are ε-close (or have statistical distance ε) if for every $A \subseteq \Omega$ it holds that $|\Pr_{x \leftarrow P}(A) - \Pr_{x \leftarrow Q}(A)| \leq \varepsilon$. By a Distribution Ensemble we mean a series $\{D_n\}_{n \in \mathbb{N}}$ where D_n is a distribution over $\{0,1\}^n$. Let $\{X_n\}$ and $\{Y_n\}$ be distribution ensembles. Define the distinguishing advantage of an algorithm A between the ensembles $\{X_n\}$ and $\{Y_n\}$ denoted as $\Delta^A(\{X_n\}, \{Y_n\})$, by:

$$\Delta^A(\{X_n\}, \{Y_n\}) = |\Pr[A(1^n, X_n) = 1] - \Pr[A(1^n, Y_n) = 1]| \quad ,$$

where the probabilities are taken over the distributions X_n and Y_n, and the randomness of A.

2.3 Family Of Pairwise-Independent Hash Functions

DEFINITION 2.1 (family of pairwise-independent hash functions)
Let $\mathcal{H}$ be a collection of functions where each function $h \in \mathcal{H}$ is from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$. $\mathcal{H}$ is an efficient family of pairwise-independent hash functions if $|h|$ (i.e., the description length of h) and $\ell(n)$ are polynomials in n, each $h \in \mathcal{H}$ is a polynomially-computable function, and for all n, for all $x \neq x' \in \{0,1\}^n$ and all $y, y' \in \{0,1\}^{\ell(n)}$,

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = y \wedge h(x') = y'] = \frac{1}{2^{2\ell(n)}}$$

There are various constructions of efficient families of pairwise-independent hash functions for any values of n and $\ell(n)$ whose description length (i.e., $|h|$) is linear in n (e.g., [CW79]). In this work we also make use of the special case in which $\ell(n) = n$ (the hash is length preserving). In such a case $\mathcal{H}$ is called an efficient family of pairwise-independent length-preserving hash functions. In some cases we cannot afford to use hash functions whose description length is linear in the input size but can afford a description that is linear in the output size. In such cases we use the following relaxation of pairwise-independent hash functions.

DEFINITION 2.2 (family of almost pairwise-independent hash functions)
Let $\mathcal{H}$ be a collection of functions where each function $h \in \mathcal{H}$ is from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$. $\mathcal{H}$ is an efficient family of δ-almost pairwise-independent hash functions if $|h|$ and $\ell(n)$ are polynomials in n, each $h \in \mathcal{H}$ is a polynomially-computable function, and for all n, for all $x \neq x' \in \{0,1\}^n$ and all $y, y' \in \{0,1\}^{\ell(n)}$,

$$\left| \Pr_{h \leftarrow \mathcal{H}}[h(x) = y \wedge h(x') = y'] - \frac{1}{2^{2\ell(n)}} \right| \leq \delta$$

Due to [CW79], [WC81] and [NN93] there exist constructions of efficient families almost pairwise-independent hash functions for any values of n, δ and $\ell(n)$ whose description length is $\mathcal{O}(log(n) + \ell(n) + log(\delta))$.

2.4 Randomness Extractors

Randomness extractors, introduced by Nisan and Zuckerman [NZ96] are an information theoretic tool for obtaining true randomness from a "weak" source of randomness. In this work, extractors are used in a computational setting to extract pseudorandomness from an imperfect source.

DEFINITION 2.3 (strong extractors)
A polynomial-time computable function $\text{Ext} : \{0,1\}^d \times \{0,1\}^n \mapsto \{0,1\}^m$ is an (explicit) (k, ε)-strong extractor if for every distribution X over $\{0,1\}^n$ with $\text{H}_\infty(X) \geq k$, the distribution $(\text{Ext}(U_d, X), U_d)$ is ε-close to (U_m, U_d).

2.5 One-way Functions

DEFINITION 2.4 (one-way functions)
Let $t : \mathbb{N} \mapsto \mathbb{N}$ and $\delta : \mathbb{N} \mapsto [0,1]$. A polynomial-time computable function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is $(t(n), \delta(n))$-one-way, if for every algorithm A of running-time at most $t(n)$

$$\Pr[A(1^n, f(U_n)) \in f^{-1}(f(U_n))] < \delta(n) \ .$$

In the case that $\delta(n) = \frac{1}{t(n)}$, we simply write that f is a $t(n)$-one-way. f is one-way if it is $p(n)$-one-way for every polynomial p, and exponentially hard if it is 2^{Cn}-one-way for some constant $C > 0$. A one-way permutation is a one-way function that is a permutation over any input length n. Finally, if f is $(t(n), 1 - \delta(n))$-one-way for $\delta > 1/\operatorname{poly}(n)$ and every polynomial t, it is customary to call f a δ-weak one-way function.

DEFINITION 2.5 (regular one-way functions)
Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a $(t(n), \delta(n))$-one-way function. f is regular if there exist a function $\alpha : \mathbb{N} \mapsto \mathbb{N}$ such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ we have

$$\left|f^{-1}(f(x))\right| = \alpha(n)$$

In the special case that α is also polynomial-time computable, f is known-regular. In this work we do not require this property and our results hold for functions with unknown-regularity. Thus, when we say regular functions we actually mean unknown-regular functions.

Few convention remarks. When the value of the security-parameter (i.e., 1^n) is clear, we allow ourselves to omit it from the adversary's parameters list. Since any one-way function is without loss of generality length-regular (i.e., inputs of same length are mapped to outputs of the same length), it can be viewed as an ensemble of functions mapping inputs of a given length to outputs of some polynomial (in the input) length. Therefore, we can write let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a one-way function, where $\ell(n)$ is some polynomial-computable function.

2.5.1 Length Preserving One-way Functions

In the following we prove the "folklore" fact that when given a one-way function it can be assumed, without loss of generality, that it is a length-preserving one. In the case that $\ell(n) < n$ one can generate a length preserving one-way function simply by padding the output with extra zeros. In the case that $\ell(n) > n$, however, one needs to be more careful in order for the input length to remain of the same order.

LEMMA 2.6
Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $(t(n), \delta(n))$-one-way function and let $\mathcal{H}$ be an efficient family of 2^{-2n}-almost pairwise-independent hash functions from $\{0,1\}^{\ell(n)}$ to $\{0,1\}^{2n}$. Define g as follows:

$$g(x_a, x_b, h) = (h(f(x_a)), h)$$

where $x_a, x_b \in \{0,1\}^n$ and $h \in \mathcal{H}$. There exists a polynomial p such that g is a length-preserving $(t(n) - p(n), \delta(n) + 2^{-n+1})$-one-way function. [5]

Proof. The function g is length preserving as both input and output are of length $2n$ plus the description of $h \in \mathcal{H}$. Let A be an algorithm that runs in time $t_A(n)$ and inverts g with probability $\varepsilon_A(n)$. Note that x_b is a dummy input (used just for padding) so the success of A is taken over (x_a, h) and A's randomness. Define B^A as follows:

[5] Since a member of $\mathcal{H}$ can be describes using $\mathcal{O}(n)$ bits, the input length of g (and therefore also its output length) is in the same order of that of f.

ALGORITHM 2.7

Algorithm B^A for inverting f.

Input: $y \in Im(f)$.

1. Choose a uniformly random $h \in \mathcal{H}$.
2. Apply $A(h(y), h)$ to get an output (x_a, x_b, h).
3. Output x_a.

Let $p(n)$ be the an upper bound on the sampling and evaluation time of h. Clearly the running-time of B^A is at most $t_A(n)+p(n)$. Algorithm B^A is sure to succeed on any choice of (y, h) so that A succeeds on $(h(y), h)$ and in addition there exists no $z \in Im(f)$ such that $h(z) = h(y)$ (h does not introduce any collision to y). Let $Col = \{(y, h) \in \{0,1\}^n \times \mathcal{H} \ : \ \exists z \in Im(f) \ : \ h(z) = h(y)\}$. Thus,

$$\begin{aligned} \Pr[B^A(f(U_n)) \in f^{-1}(f(U_n))] &\geq \Pr[A(g(U_n, H)) \in g^{-1}(g(U_n, H)) \bigwedge g(U_n, H) \notin Col] \\ &\geq \Pr[A(g(U_n, H)) \in g^{-1}(g(U_n, H))] - \Pr[g(U_n, H) \in Col] \ , \end{aligned}$$

where H is a random variable uniformly distributed over $\mathcal{H}$. For all $y, z \in Im(f)$ the almost pairwise-independence of h yields that the probability that $H(z) = H(y)$ is at most 2^{-2n+1}. Since the size of $Im(f)$ in $\{0,1\}^{\ell(n)}$ is at most 2^{n+1},a union bound over all possible $z \in Im(f)$ gives that for any $y \in Im(f)$ it holds that $\Pr[\exists z \in Im(f) : H(z) = H(y)] \leq 2^{-n+1}$. Therefore, by an averaging argument $\Pr[(f(U_n), H) \in Col] \leq 2^{-n+1}$. Putting it all together we get that $\Pr[B^A(f(U_n)) \in f^{-1}(f(U_n))] \geq \varepsilon_A(n) - 2^{-n+1}$. □

2.6 Hardcore Predicates and Functions

Hard-core predicates/functions have a major role in the construction of pseudorandom generators based on one-way functions.

DEFINITION 2.8 (hardcore functions)

Let t and δ be functions from $\mathbb{N}$ to $\mathbb{N}$ and from $\mathbb{N}$ to $[0,1]$ respectively, let $L_n \subseteq \{0,1\}^n$ and let $f : \{0,1\}^n \mapsto \{0,1\}^*$ and $hc : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be two functions defined over $\{0,1\}^n$. We call hc a $(t(n), \delta(n))$-hardcore function of f over L_n, if it is polynomial-time computable and for every algorithm A with running time at most $t(n)$

$$\Delta^A\left((f(x), hc(x))_{x \leftarrow L_n}, (f(x), U_{\ell(n)})_{x \leftarrow L_n}\right) < \delta(n) \ ,$$

for all but finitely many n's.

In the case that $\delta(n) = \frac{1}{t(n)}$, we simply say that hc is a $t(n)$-hardcore function. If hc is $p(n)$-hardcore function for every polynomial p, then we call it an hardcore function of f. As a convention in the case that $L_n = \{0,1\}^n$, it is omitted it form the definition of hc. If hc is a predicate (i.e., $\ell(n) = 1$), then it is called an hardcore predicate of f. Finally, it is custom to call the value $hc(x)$, the "hardcore-bits" of $f(x)$.

The following theorem is an immediate generalization of the Goldreich-Levin hardcore function [GL89, Corollary 1].

THEOREM 2.9
There exists an efficiently computable family of functions $\{gl_i : \{0,1\}^{3n} \mapsto \{0,1\}^i\}_{i\in[n]}$, a constant $c \in (0,1)$ and a polynomial p such that the following holds. Let $L_n \subseteq \{0,1\}^n$, let f and g be two polynomial-time computable functions from $\{0,1\}^n$ to $\{0,1\}^n$ and assuming that for every algorithm A with running time at most $t(n)$ it holds that

$$\Pr_{x\leftarrow L_n}[A(f(x)) = g(x)] \leq \frac{1}{t(n)} \ .$$

Then the following holds for every function t' satisfying $p(n, t'(n)) < t(n)$. For every value of $\ell \in \{1, \ldots, \lfloor c \log(t(n)) \rfloor\}$, the function $hc : \{0,1\}^{3n} \mapsto \{0,1\}^\ell$ defined as $hc(x,r) = gl_\ell(g(x), r)$, is a $t'(n)$-hardcore function of $f'(x,r) = (f(x), r)$ over L_n.

2.7 A Uniform Extraction Lemma

The following lemma is a generalization of the (uniform version) of Yao's XOR lemma. Given m independent $(t, 1-\delta)$-hardcore bits, we would like to extract approximately δm pseudorandom bits out of them. The version we present here generalizes [HILL99, Lemma 6.5]), in particular the original lemma required the hardcore predicate to have a hardcore-set (i.e., a subset of inputs such that the value of the predicate is unpredictable [computationally] over this subset), where in the following lemma this property is no longer required. In addition, the original lemma was tailored for the specific function and predicate it was used with, where the following lemma suits any hard predicate. Finally, the original lemma is stated using an efficient family of pairwise-independent hash functions, while the following lemma is stated using any explicit randomness extractor. The lemma is proven using Holenstein's "uniform hardcore lemma" [Hol05]. [6]

LEMMA 2.10
Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a polynomial-time computable function, let b be a $(t(n), 1-\delta(n))$-hardcore predicate of f and let $\mathrm{Ext} : \{0,1\}^m \times \{0,1\}^d \mapsto \{0,1\}^r$ be a (k,ε)-strong-extractor. We define $hc : \{0,1\}^{mn} \times \{0,1\}^d \mapsto \{0,1\}^r$ as $hc(x_1, \ldots, x_m, y) = \mathrm{Ext}(y, b(x_1), \ldots, b(x_m))$ and let $f'(x_1, \ldots, x_m, y) = (f(x_1), \ldots, f(x_m), y)$.

There exists a polynomial p such that the following holds. For any $\gamma(n) > m \cdot 2^{-n/4}$ and $t'(n)$ satisfying $p(t'(n), 1/\gamma(n), 1/\delta(n), m, n) < t(n)$, the function hc is a $(t'(n), \varepsilon + \rho(n) + \gamma(n))$-hardcore function of f', where $\rho(n)$ is the probability that when taking m independent samples in $\{0,1\}^n$ less than k samples are in some fixed subset of density $\delta(n)$.

Proof. For clarity we omit the value n whenever it is clear from the context. Let A be an algorithm that runs in time t_A and given $f'(x_1, \ldots, x_m, y)$, distinguishes the value of $hc(x_1, \ldots, x_m, y)$ from random with advantage $\varepsilon_A > \varepsilon + \rho + \gamma$. For a fixed set $S \subseteq \{0,1\}^n$ of density δ, define the following,

[6] Independently of this work, [Hol06b, Theorem 7.3] presents a different version of Lemma 2.10. While their theorem yields some generalization over the following lemma (informally it also considers the situation where some information about the hardcore bits leaks to the adversary), it is only stated for the case of polynomial hardness.

not necessarily efficiently computable, randomized predicate $Q : \{0,1\}^n \mapsto \{0,1\}$.

$$Q(x) = \begin{cases} U_1 & x \in S, \\ b(x) & \text{otherwise.} \end{cases} \tag{1}$$

For any $i \in \{0, \ldots, m\}$ we define the distribution D^i as:

$$D^i = (f(U_n^1), \ldots, f(U_n^1), U_d, \text{Ext}(U_d, b(U_n^1), \ldots, b(U_n^i), Q(U_n^{i+1}), \ldots, Q(U_n^m)).$$

We first note that D^m is equal to $(f'(U_n^1, \ldots, U_n^m, U_d), hc(U_n^1, \ldots, U_n^m, U_d))$ (i.e., to the output of f' concatenated with our candidate hardcore function hc). Also, since with probability $1 - \rho$ the min-entropy of D^0 is k, then by the properties of the extractor we have that the statistical distance between D^0 and $(f(U_n^1), \ldots, f(U_n^1), U_d, U_r)$ is at most $\varepsilon + \rho$. It follows that A distinguishes between D^0 and D^m with advantage $\varepsilon_A - \rho - \epsilon > \gamma$. Hence, by a standard hybrid argument we deduce that there exists a $j \in \{0, \ldots, m-1\}$ such that A distinguishes between D^j and D^{j+1} with probability γ/m. Moreover, Since D^j and D^{j+1} are identical when $x_{j+1} \notin S$, it follows that A must achieve this distinguishing probability between D^j and D^{j+1} also when given that $x_{j+1} \in S$. Finally, note that the only difference between D^j and D^{j+1} given that $x_{j+1} \in S$ is whether the $(j+1)$'th input to Ext is $b(x_j)$ or a random input. The following algorithm given oracle access to the characteristic function χ^S of any set of density δ, returns, with probability $1 - 2^{-n}$, a circuit that distinguishes between $(f(x), b(x))_{x \leftarrow S}$ and $(f(x), U)_{x \leftarrow S}$ with probability γ/m.

ALGORITHM 2.11

Algorithm B^{χ^S}.

1. Find a $j \in \{0, \ldots, m-1\}$ such that, with probability at least $1 - 2^{-n}$, A distinguishes between D^j and D^{j+1} with success probability at least γ/m. This can be done using χ^S, by sampling form distributions D^j and D^{j+1} and running A on them.[7]
2. Sample d^j, an instance of D^j (again using χ^S).
3. Return the circuit C that on input (y, b) replaces $f(x_{j+1})$ and $b(x_{j+1})$ in d^j by y and b respectively and outputs $A(d^j)$.

Note that t_B, the running-time of B, is polynomial in t_A, δ, m, δ and n. For any fixed $S \subseteq \{0,1\}^n$ of density δ let $adv_S = \mathsf{Ex}[C \leftarrow B^{\chi^S} : \Delta^C((X_n, b(X_n)), (f(X_n), U))]$, where X_n is uniformly distributed over S_n and the expectation is over the random coins of B. By the above observations it follows that $adv_S > \gamma - 2^{-n} > 2^{-n/3}$. By [Hol06a, Propostition 3] there exists an algorithm that runs in time polynomial in n, m, t_B and $1/adv_S$, and distinguishes between $(f(U_n), b(U_n))$ and $(f(U_n), U)$ with probability at least $1 - \delta$. [8] Thus, choosing p to accommodate the running time of B and of the algorithm guaranteed by [Hol06a, Propostition 3], the hardness of b yields that $p(t_A, 1/\gamma, 1/\delta, m, n) \geq t(n)$. □

[7] Alternatively, one can just pick a random $j \in \{0, \ldots, m-1\}$ and succeed with probability at least $\frac{1}{m}$.

[8] The original form of the uniform hardcore lemma, [Hol05, Lemma 2.5], was stated only for super polynomial hardness and [Hol06a, Propostition 3] generalizes it for any hardness.

2.8 Pseudorandom Generators

DEFINITION 2.12 (pseudorandom generators)
Let t and δ be functions from $\mathbb{N}$ to $\mathbb{N}$ and to $[0,1]$ respectively, and let $G : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a polynomial-time computable function where $\ell(n) > n$. We say that G is a $(t(n), \delta(n))$-pseudorandom-generator if for every algorithm A with running time at most $t(n)$

$$\Delta^A \left(G(U_n), U_{\ell(n)} \right) < \delta(n) \ .$$

In the case that $\delta(n) = \frac{1}{t(n)}$, we simply say that G is a $t(n)$-pseudorandom-generator. G is a pseudorandom-generator if it is $p(n)$-pseudorandom-generator for every polynomial p.

2.9 Bounded-Space Generators

Bounded-Space generators are pseudorandom generators against bounded-space adversaries. Such generators plays a central role in derandomization tasks. We are interested in generator for the following type of adversaries.

DEFINITION 2.13 (bounded-width layered branching program - LBP)
A (s,t,v)-LBP M is a directed graph with $2^s \cdot (t+1)$ vertices, partitioned into $t+1$ layers with 2^s vertices in each layer. Each vertex in the i'th layer has exactly 2^v outgoing labeled edges to the $(i+1)^{st}$ layer, one for every possible string $z \in \{0,1\}^v$. The vertices in the last layer (the t layer) are labeled by 0 or 1.

Denote by M_x such a LBP with starting vertex $x \in \{1, \ldots, 2^s\}$ in the 0'th level. For a sequence $\overline{z} \in \{0,1\}^{tv}$, we define the output of the LBP $M_x(\overline{z})$ by a walk on the graph starting at vertex x in layer 0 and advancing to the i'th layer along the edge labeled by $\overline{z}_i$. $M_x(\overline{z})$ accepts if it reaches a vertex labeled by 1 and rejects otherwise.

DEFINITION 2.14
A generator $G : \{0,1\}^m \mapsto \{0,1\}^{tv}$ is said to ε-fool a LBP M if for every $x \in \{1, \ldots, 2^s\}$ we have:

$$|[M_x(U_{tv}) \text{ accepts}] - \Pr[M_x(G(U_m)) \text{ accepts}]| < \varepsilon$$

THEOREM 2.15 [Nis92, INW94]
For every s,t,v there exist a generator $BSG : \{0,1\}^{\mathcal{O}(v+(s+\log(\frac{t}{\varepsilon}))\log t)} \mapsto \{0,1\}^{tv}$ running in time $poly(s,t,v)$ that ε-fools every (s,t,v)-LBP.

2.10 The Security of Cryptographic Constructions

Typically the proof of security for cryptographic constructions is based on reductions. In this paradigm we use a presumably secure implementation of one primitive (or possibly several primitives) in order to implement a second primitive. The proof of security for the second primitive relies on the security assumption for the original one. More precisely, we prove that any efficient adversary that breaks the implementation of the second primitive can be used to efficiently break the original primitive. Note that the meaning of "breaking a primitive" and, furthermore, the definition of the *success probability* of an adversary in breaking the primitive, varies between different

primitives. For example, in the case of one-way functions the success probability is the fraction of inputs on which the adversary manages to invert the function. Usually, there is a tradeoff between the running-time of an adversary and its success probability (e.g., it may be possible to utterly break a primitive by enumerating all possibilities for the secret key). Therefore, both the running-time and success probability of possible adversaries are relevant when analyzing the security of a primitive. A useful, combined parameter is the *time-success ratio* of an adversary which we define next.

DEFINITION 2.16 (time-success ratio)
Let P be a primitive and let A be an adversary running in time $t_A(n)$ and breaking P with probability $\varepsilon_A(n)$. The time-success ratio of A in breaking P is defined as $R(n) = \frac{t_A(n)}{\varepsilon_A(n)}$, where n is the security-parameter of the primitive.[9]

Note that in the above definition, the smaller the R the better A is in breaking P. A quantitative analysis of the security of a reduction is crucial for both theoretical and practical reasons. Given an implementation of primitive P using primitive Q along with a proof of security, let R_P be the security-ratio of a given adversary w.r.t. P and let R_Q be the security-ratio of the adversary that the proof of security yields. A natural way to measure the security of a reduction is by the relation between R_P and R_Q. Clearly, the smaller the R_Q comparing to R_P, the better the performance of the adversary the reduction yields when trying to break Q comparing to the performance of the adversary trying to break P.

The most desirable reductions is when $R_Q(n) \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n))$. In such reductions, known as *linear-preserving reductions*, we are guaranteed that breaking the constructed primitive is essentially as hard as breaking the original one. Next we find the *polynomial-preserving reductions* when $R_Q \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n)^{\mathcal{O}(1)})$. Note that a linear/polynomial-preserving reduction typically means that the inputs of Q and P are of the same length (up to a constant-ratio). The other side of the scale is when $R_Q \in n^{\mathcal{O}(1)}\mathcal{O}(R_P(n^{\mathcal{O}(1)}))$. In such reductions, known as *weak-preserving reductions*, we are only guaranteed that breaking P is as hard as breaking Q for polynomially smaller security-parameter (e.g., polynomially smaller input length). For a more comprehensive discussion about the above issues the reader may refer to [HL92]. This quantitative classification of security preserving reduction partly motivates our focus on the input-length as the main parameter that our reductions aim to improve. In particular, better space bounded generators would make our reduction polynomial-preserving rather than weak-preserving (see Section 2.9).

3 Pseudorandom Generators from Regular One-way Functions

The following discussion considers only length preserving regular one-way functions. Note that by Section 3.4.1, this is without lost of generality.

[9] It is convenient to define the security-parameter of a primitive as its input length. This is in particular the convention for the primitives discussed in this work.

3.1 Some Motivation and the Randomized Iterate

Recall that the BMY generator simply iterates the one-way permutation f on itself, and outputs a hardcore-bit of the intermediate step at each iteration. The crucial point is that the output of the function is also uniform in $\{0,1\}^n$ since f is a permutation. Hence, when applying f to the output, it is hard to invert this last application of f, and therefore hard to predict the new hardcore-bit (Yao shows [Yao82] that the unpredictability of bits implies pseudorandomness). Since the seed is essentially just an n bit string and the output is as long as the number of iterations, then the generator actually stretches the seed.

We want to duplicate this approach for general one-way functions, unfortunately the situation changes drastically when the function f is not a permutation. After a single application of f, the output may be very far from uniform, and in fact may be concentrated on a very small and easy fraction of the inputs to f. Thus reapplying f to this output gives no hardness guarantees at all. In an attempt to salvage the BMY framework, Goldreich et. al. [GKL93] suggested to add a randomization step between every two applications of f, thus making the next input to f a truly random one. This modification that we call randomized iterates lies at the core of our work and is defined next.

DEFINITION 3.1 (the randomized iterate)
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$, let $\mathcal{H}$ be an efficient family of pairwise-independent length-preserving hash functions defined over $\{0,1\}^n$ and let $\ell \in \mathbb{N}$. For $k \in [\ell]$, $x \in \{0,1\}^n$ and $\overline{h} \in \mathcal{H}^\ell$ define the k'th randomized iterate $f^k : \{0,1\}^n \times \mathcal{H}^\ell \mapsto Im(f)$ recursively as

$$f^k(x,\overline{h}) = f(\overline{h}_k(f^{k-1}(x,\overline{h}))) \ ,$$

where $f^0(x,\overline{h}) = f(x)$. In the following we denote by H^k the random variable uniformly distributed over $\mathcal{H}^k$.

The application of the randomized iterate for pseudorandom generators is a bit tricky. On the one hand, such a randomization costs a large number of random bits, much larger than what can be compensated for by the hardcore-bits generated in each iteration. So in order for the output to actually be longer than the input, we also output the descriptions of the hash functions. But on the other hand, handing out the randomizing hash gives information on intermediate values such as $f^i(x,\overline{h})$, and f might no longer be hard to invert when applied to such an input. Somewhat surprisingly, the randomized iterate of a regular one-way function remains hard to invert even when the hash functions are known. This fact, which is central to the whole approach, was proved in [GKL93] when using a family of n-wise independent hash functions. As a first step, we give a simpler proof that extends to pairwise-independent hash functions as well.

REMARK 3.2
In the definition randomized iterate we define $f^0(x,\overline{h}) = f(x)$. This was chosen for ease of notation and for consistency with the results for general one-way functions (see Section 5). For the regular one-way function construction it suffices to define $f^0(x,\overline{h}) = x$, thus saving a single application of the function f.

3.2 The Last Randomized Iteration is Hard to Invert

In this section we formally state and prove the key observation mentioned above. After applying k randomized-iterations of a regular one-way function f, it is hard to invert the last-iteration, even if given access to all of the hash functions leading up to this point.

LEMMA 3.3

Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular $(t(n), \delta(n))$-one-way, let $k \in \text{poly}(n)$, and let f^k and $\mathcal{H}$ be as in Definition 3.1. Then there exists a polynomial p, such that for every algorithm A of running-time at most $(t(n) - p(n))$ it holds that

$$\Pr[A(f^k(U_n, H^k), H^k) = f^{k-1}(U_n, H^k)] \leq \sqrt[3]{8(k+1)\delta(n)}$$

for large enough n, where the probability is also taken over the random coins of A.

We briefly give some intuition to the proof, illustrated with regard to the first randomized iterate. Suppose that we have an algorithm A that *always* finds $f^0(x,h) = f(x)$ given $f^1(x,h)$ and h. In order to invert the one-way function f on an element $z \in Im(f)$, we simply need to find a hash h' that is consistent with z, in the sense that there exists an x' such that $z = f^1(x', h')$. Now we simply run $y = A(z, h')$, and output $h'(y)$ (and indeed $f(h'(y)) = z$). The point is that if f is a regular function, then finding a consistent hash is easy, simply because a random and independent h' is likely to be consistent with z. The actual proof follows this framework, but is far more involved due to the fact that the reduction starts with an algorithm A that has only a small (yet polynomial) success probability.

Proof. Let A be an algorithm with running-time $t_A(n)$ such that

$$\Pr[A(f^k(U_n, H^k), H^k)) = f^{k-1}(U_n, H^k)] \geq \varepsilon_A(n) \ ,$$

where $\varepsilon_A(n) > \sqrt[3]{8(k+1)\delta(n)}$. We show that unless t_A is large, we can use A in order to violate the hardness of f. Consider the procedure M^A for this task.

ALGORITHM 3.4

Algorithm M^A for inverting f.
Input: $y \in Im(f)$.

1. Choose uniformly at random $\overline{h} \in \mathcal{H}^k$.
2. Apply $A(y, \overline{h})$ to get an output x.
3. Output $\overline{h}_k(x)$.

Letting $p(n)$ be the sampling and evaluation time of $\overline{h}$, we have that the running-time of M^A is at most $t(n)$. The rest of the proof of Lemma 3.3 shows that M^A succeeds with probability at least $\varepsilon_A(n)^3/8(k+1) > \delta(n)$ on uniformly chosen $y \in Im(f)$, and we conclude that $t_A(n) > t(n) - p(n)$.

We start by focusing our attention only on those inputs for which A succeeds reasonably well. Recall that the success probability of A is taken over the choice of inputs to A as induced by the choice of $x \in \{0,1\}^n$ and $\overline{h} \in \mathcal{H}^k$ and the internal coin-tosses of A. The following Markov argument implies that the probability of getting an element in the set that A succeeds on is not very small.

CLAIM 3.5
Let $S_A \subseteq Im(f^k) \times \mathcal{H}^k$ be the subset defined as

$$S_A = \left\{ (y, \overline{h}) \in Im(f^k) \times \mathcal{H}^k : \Pr[f(\overline{h}_k(A(y, \overline{h})) = y] > \varepsilon_A(n)/2 \right\} \ ,$$

then

$$\Pr[(f^k(U_n, H^k), H^k) \in S_A] \geq \varepsilon_A(n)/2 \ .$$

Proof. Suppose that $\Pr[(f^k(U_n, H^k), H^k) \in S_A] < \varepsilon_A(n)/2$. Then we have:

$$\begin{aligned} & \Pr[A(f^k(U_n, H^k), H^k) = f^{k-1}(U_n, H^k)] \\ < \ & \frac{\varepsilon_A(n)}{2} \cdot \Pr[(f^k(U_n, H^k), H^k) \notin S_A] + 1 \cdot \Pr[(f^k(U_n, H^k), H^k) \in S_A] \\ < \ & \frac{\varepsilon_A(n)}{2} + \frac{\varepsilon_A(n)}{2} = \varepsilon_A(n) \ . \end{aligned}$$

which contradicts the assumption about the success probability of A. □

Now that we identified a heavy subset of the inputs that A succeeds upon, we want to say that M^A has a fair chance to hit outputs induced by this subset. This is formally shown in the following lemma.

LEMMA 3.6
For every set $T \subseteq Im(f^k) \times \mathcal{H}^k$, if

$$\Pr[(f^k(U_n, H^k), H^k) \in T] \geq \gamma \ ,$$

then

$$\Pr[(f(U_n), H^k) \in T] \geq \frac{\gamma^2}{k+1} \ .$$

Assuming Lemma 3.6 we may conclude the proof of Lemma 3.3. Claim 3.5 yields that $\Pr[(f^k(U_n, H^k), H^k) \in S_A] \geq \frac{\varepsilon_A(n)}{2}$. By Lemma 3.6 taking $T = S_A$ and $\gamma = \varepsilon_A(n)/2$ we get that $\Pr[(f(U_n), H^k) \in S_A] \geq \varepsilon_A(n)^2/4(k+1)$. Thus M^A has a $\varepsilon_A(n)^2/4(k+1)$ chance of hitting the set S_A on which it will succeed with probability at least $\varepsilon_A(n)/2$. Altogether, M^A succeeds in inverting f with probability $\varepsilon_A(n)^3/8(k+1) > \delta(n)$. □

Proof. (of Lemma 3.6) The lemma essentially states that with respect to $\widetilde{f^k}(x, \overline{h}) = (f^k(x, \overline{h}), \overline{h})$ (i.e, the function defined by concatenating the input hash functions to the output of f^k), any large subset of inputs induces a large subset of outputs. Thus, there is a fairly high probability of hitting this output set simply by sampling independent y and $\overline{h}$. Intuitively, if a large set of inputs induces a small set of outputs, then there must be many collisions in this set (a collision means that two different inputs lead to the same output). We show that this is impossible, however, by proving that the collision-probability of the function $(f^k(x, \overline{h}), \overline{h})$ is small.

CLAIM 3.7

$$\mathrm{CP}((f^k(U_n, H^k), H^k) \leq \frac{k+1}{|\mathcal{H}|^k \cdot |Im(f)|}$$

Proof. For every two inputs $(x_0, \overline{h_0})$ and $(x_1, \overline{h_1})$ to f^k, in order to have a collision we must first have that $\overline{h_0} = \overline{h_1}$, which happens with probability $(1/|\mathcal{H}|)^k$. Now, given that $\overline{h_0} = \overline{h_1} = \overline{h}$ (with a random $\overline{h} \in \mathcal{H}^k$), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$. If $f(x_0) = f(x_1)$ (happens with probability $1/|Im(f)|$) then a collision is assured. Otherwise, there must be an $i \in [k]$ for which $f^{i-1}(x_0, \overline{h}) \neq f^{i-1}(x_1, \overline{h})$ but $f^i(x_0, \overline{h}) = f^i(x_1, \overline{h})$. Since $f^{i-1}(x_0, \overline{h}) \neq f^{i-1}(x_1, \overline{h})$, due to the pairwise-independence of h_i, the values $h_i(f^{i-1}(x_0, \overline{h}))$ and $h_i(f^{i-1}(x_1, \overline{h}))$ are uniformly random values in $\{0,1\}^n$, and thus $f(h_i(f^{i-1}(x_0, \overline{h}))) = f(h_i(f^{i-1}(x_1, \overline{h})))$ happens with probability $1/|Im(f)|$. Altogether, $\mathrm{CP}((f^k(U_n, \mathcal{H}^k), \mathcal{H}^k) \leq \frac{1}{|\mathcal{H}|^k} \sum_{i=0}^{k} \frac{1}{|Im(f)|} = \frac{k+1}{|\mathcal{H}|^k \cdot |Im(f)|}$. □

On the other hand, we check the probability of getting a collision inside the set T, which is a lower bound on the probability of getting a collision at all. We first request that both $(x_0, \overline{h_0}) \in T$ and $(x_1, \overline{h_1}) \in T$. This happens with probability at least γ^2. Then, once inside T, we know that the probability of collision is at least $1/|T|$. Altogether:

$$\mathrm{CP}(f^k(U_n, H^k), H^k) \geq \gamma^2 \cdot \frac{1}{|T|} \quad . \tag{2}$$

Combining Claim 3.7 and (2), we get $\frac{|T|}{|\mathcal{H}|^k \cdot |Im(f)|} \geq \frac{\gamma^2}{k+1}$. Since the probability of getting a value in T when choosing a random element in $Im(f) \times \mathcal{H}^k$ is exactly $\frac{|T|}{|\mathcal{H}|^k |Im(f)|}$ (this follows since for a regular function f the distribution $f(U_n)$ is the uniform distribution over $Im(f)$). It follows that $\Pr[(y, \overline{h}) \in T] \geq \gamma^2/(k+1)$ as requested. □

REMARK 3.8

The last fact in the proof of Lemma 3.6 (that $f(U_n)$ is uniformly distributed over $Im(f)$) is in fact the only place where the regularity of the one-way function is required. The proof fails for general one-way functions where the preimage size of different elements may vary considerably. For a general function, the collision-probability argument at the heart of this lemma can be made to hold only as long as the last element in a sequence of applications is at least as heavy as all the elements along the sequence (see Lemma 4.1). While for regular functions this requirement is always true, for general one-way functions this occurs with probability that deteriorates linearly (in the length of the sequence). Thus using a long sequence of iterations is likely to lose the hardness of the original one-way function (see more in Section 4).

3.3 A Pseudorandom Generator from a Regular One-way Function

After showing that the randomized-iterations of a regular one-way function are hard to invert, it is natural to follow the footsteps of the BMY construction to construct a pseudorandom generator. Rather than using simple iterations of the function f, randomized-iterations of f are used

instead, with fresh randomness in each application. As in the BMY case, a hardcore-bit(s) of the current input is taken at each stage. In order to keep things more readable, we start by giving our pseudorandom generator based on regular one-way functions with super polynomial hardness (i.e., standard one-way functions). In Section 3.3.1 generalize this result to regular one-way functions with arbitrary hardness. In particular, we get more efficient pseudorandom generators, assuming that underlying regular one-way functions are exponentially hard.

THEOREM 3.9

Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular one-way function and let $\mathcal{H}$ be an efficient family of pairwise-independent length preserving hash functions. Define $G : \{0,1\}^n \times \mathcal{H}^n \times \{0,1\}^{2n} \mapsto \{0,1\}^{n+1} \times \mathcal{H}^n \times \{0,1\}^{2n}$ as

$$G(x, \overline{h}, r) = (b(f^0(x, \overline{h}), r), \ldots, b(f^n(x, \overline{h}), r), \overline{h}, r),$$

where:

- b is the Goldreich-Levin hardcore predicate (see Theorem 2.9),
- Recall that $f^0(x, \overline{h}) = f(x)$ and for $i \in [n]$ it holds that $f^i(x, \overline{h}) = f(\overline{h}_i(f^{i-1}(x, \overline{h})))$.

Then, G is a pseudorandom generator.

Proof. Lemma 3.3 yields that no efficient algorithm can compute f^{k-1} given $(f^k, h_1, \ldots, h_k)$ with non-negligible success probability. Hence, Theorem 2.9 guarantees that $b(f^{k-1})$ is a hardcore function of $f'(x, h_1, \ldots, h_k) = (f^k(x, h_1, \ldots, h_k), h_1, \ldots, h_k)$. In the following we show that any algorithm that breaks the pseudorandomness of G too well, violates the hardness of $b(f^{k-1})$ for some $k \in [n]$.

Yao [Yao82] showed using a hybrid argument that it is, up to linear factor, as hard to distinguish a pseudorandom sequence from a random one, as it is to predict the next bit of the sequence for every prefix of the sequence. Since it is as hard to distinguish the sequence $(r, \overline{h}, b(f^n), \ldots, b(f^0))$ from random as it is to distinguish the output of G (for ease of notation we omit the input $(x, \overline{h})$ to the f^k's and the input r from the hardcore bits), by [Yao82] it suffices to show that for every $k \in [n]$ it is hard to predict $b(f^{k-1})$ given $(r, \overline{h}, b(f^n), \ldots, b(f^k))$. Assume toward a contradiction the existence of an efficient algorithm A for which $\Pr[A(r, \overline{h}, b(f^n), \ldots, b(f^k)) = b(f^{k-1})] > \frac{1}{2} + \mathrm{neg}(n)$. Consider the following efficient algorithm M^A for predicting $b(f^{k-1})$ given $(r, \overline{h}, f^k(x, \overline{h}))$.

ALGORITHM 3.10

Predictor M^A.
Input: $(r, h_1, \ldots, h_k, f^k(x, h_1, \ldots, h_k))$.

1. Choose uniformly at random $h_{k+1}, \ldots, h_n \in \mathcal{H}$,
2. Generate $f^{k+1}, \ldots, f^n$ from $(f^k, h_{k+1}, \ldots, h_n)$.
3. Output $A(r, h_1 \ldots, h_n, b(f^n), \ldots, b(f^{k+1}), b(f^k))$

By choosing $h_{k+1}, \ldots, h_n$ independently at random, M^A generates a series $(f^1, \ldots, f^n)$ that has the same distribution as in the evaluation of G. Thus, the procedure M^A succeeds in predicting $b(f^{k-1})$ with probability at least $\frac{1}{2} + \mathrm{neg}(n)$, and a contradiction is derived. □

3.3.1 From any hardness

The next theorem generalizes Theorem 3.9 for regular one-way function with arbitrary hardness. Since it follows the same lines as the proof of Theorem 3.9, the proof of the next theorem is omitted.

THEOREM 3.11
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular $t(n)$-one-way function and let $\mathcal{H}$ be an efficient family of pairwise-independent length preserving hash functions. Define $G : \{0,1\}^n \times \mathcal{H}^d \times \{0,1\}^{2n} \mapsto \{0,1\}^{(d+1)\ell}\mathcal{H}^n \times \{0,1\}^{2n}$ as

$$G(x, \overline{h}, r) = (gl_\ell(f^0(x, \overline{h}), r), \ldots, gl_\ell(f^d(x, \overline{h}), r), \overline{h}, r),$$

where:

- $\ell = \left\lfloor \frac{c}{4} \cdot \log(t(n)) \right\rfloor$ and $d = \lceil n/\ell \rceil + 1$, where c is the constant that appears in Theorem 2.9, [10]
- gl_ℓ is the Goldreich-Levin hardcore function (see Theorem 2.9),
- Recall that $f^0(x, \overline{h}) = f(x)$ and for $i \in [d]$ it holds that $f^i(x, \overline{h}) = f(\overline{h}_i(f^{i-1}(x, \overline{h})))$.

There exists a polynomial p such that G is a $t'(n)$-pseudorandom generator for any t' satisfying $p(n, t'(n)) < t(n)$. The input length of G is $\mathcal{O}(\frac{n^2}{\log t(n)})$ and it stretches its input by $\Omega(\log t(n))$.

3.4 An Almost-Linear-Input Construction from a Regular One-way Function

Assuming that the underlying function is one-way in the usual sense (i.e., of super-polynomial hardness), the pseudorandom generator presented in the previous section (when using Theorem 3.11) stretches a seed of length $\mathcal{O}(n^2/\log(n))$ by $\log(n)$ bits. Although this is an improvement over the GKL generator, it still translates to a rather high loss of security, since the security of the generator on m bits relies on the security of regular one-way function on $\sqrt{m}$ bits. In this section we give a modified construction of the pseudorandom generator that takes a seed of length only $m = \mathcal{O}(n \log n)$.

Notice that the input length of the generator of Theorem 3.11 is dominated by the description of the d independent hash functions $\overline{h} = (h_1, \ldots, h_d)$. The idea of the new construction is to give a derandomization of the choice of the d hash functions. Thus $h_1, \ldots, h_d$ are no longer chosen independently, but are chosen in a way that is sufficient for the proof to go through. The derandomization uses generators against bounded-space distinguishers, and specifically we can use the generator of Nisan [Nis92], (or that of Impagliazzo, Nisan and Wigderson [INW94]). An important observation is that calculating the randomized iterate of an input can be viewed as a bounded-space algorithm, alternatively presented here as a bounded-width layered branching-program. More accurately, at each step the branching program gets a random input h_i and produces $f^{i+1} = f(h_i(f^i))$. We will show that indeed when replacing $h_1, \ldots, h_d$ with the output of a generator that fools related branching programs, then the proof of security still holds (and specifically the proof of Lemma 3.6).

THEOREM 3.12
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a regular $t(n)$-one-way function and let $\mathcal{H}$ be an efficient family of pairwise-independent length preserving hash functions. Let $v(\mathcal{H})$ be the description length of $h \in H$

[10] If f is a one-way function in the usual sense (i.e., of super-polynomial hardness), we let $\ell = \lfloor \log(n) \rfloor$.

and let $BSG : \{0,1\}^{\tilde{n}\in\mathcal{O}(n\log n)} \mapsto \{0,1\}^{nv(\mathcal{H})}$ be a bounded-space generator that 2^{-n}-fools every $(2n, n, v(\mathcal{H}))$-LBP. [11] We define $G' : \{0,1\}^n \times \{0,1\}^{\tilde{n}} \times \{0,1\}^{2n} \mapsto \{0,1\}^{(d+1)\ell+\tilde{n}+2n}$ as

$$G'(x, s, r) = (gl_\ell(f^0(x, BSG(s)), r), \ldots, gl_\ell(f^d(x, BSG(s)), r), \overline{h}, r),$$

where:

- $\ell = \left\lfloor \frac{c}{4} \cdot \log(t(n)) \right\rfloor$ and $d = \lceil n/\ell \rceil$, where c is the constant that appears in Theorem 2.9,
- gl_ℓ is the hardcore function of Theorem 2.9,
- $BSG(s) = \overline{h} \in \mathcal{H}^d$,
- Recall that $f^0(x, \overline{h}) = f(x)$ and for $1 \leq i \leq d$ $f^i(x, \overline{h}) = f(\overline{h}_i(f^{i-1}(x, \overline{h})))$.

Then there exists a polynomial p that for any t' satisfying $p(n, t'(n)) < t(n)$, it holds that G' is a $t'(n)$-pseudorandom generator.

Proof outline. The proof of the derandomized version follows in the steps of the proof of Theorem 3.11. We give a high-level outline of this proof, focusing only on the main technical lemma that changes slightly. The proof first shows that given the k'th randomized iterate $f^k(x, \overline{h})$ and $\overline{h}$ it is hard to compute $f^{k-1}(x, \overline{h})$ (analogously to Lemma 3.3), only now this also holds when the hash functions are chosen as the output of the bounded-space generator. The proof is identical to the proof of 3.3, only replacing appearances of $\overline{h}$ with the seed s. Again, the key to the proof is the following technical lemma (slightly modified from Lemma 3.6).

LEMMA 3.13

For every set $T \subseteq Im(f) \times \{0,1\}^{\tilde{n}}$, if

$$\Pr[(f^k(U_n, BSG(U_{\tilde{n}})), U_{\tilde{n}}) \in T] \geq \gamma \ ,$$

then

$$\Pr[(f(U_n), U_{\tilde{n}}) \in T] \geq \gamma^2/(k+2) \ .$$

Once we know that $f^{k-1}(x, BSG(s))$ is hard to compute given $f^k(x, BSG(s))$ and s, we deduce that one cannot predict a hardcore-bit $gl_\ell(f^{k-1}(x, BSG(s)), r)$ given $f^k(x, BSG(s))$ and the seed s to the bounded-space generator. From here, the proof follow just as the proof of Theorem 3.11 in showing that the output of G' is an unpredictable sequence and therefore a pseudorandom sequence.

Proof. (of Lemma 3.13) Denote by $g : \{0,1\}^n \times \{0,1\}^{\tilde{n}} \mapsto Im(f) \times \{0,1\}^{\tilde{n}}$ the function taking inputs of the form (x, s) to outputs of the form $(f^k(x, BSG(s)), s)$. We proceed by giving bounds on the collision-probability of g. For every two inputs to g, $(x_0, \tilde{h_0})$ and $(x_1, \tilde{h_1})$, in order to have a collision we must first have that $\tilde{h_0} = \tilde{h_1}$ which happens with probability $1/2^{\tilde{n}}$. Now, given that $\tilde{h_0} = \tilde{h_1} = s$ (with a random s), we analyze the probability of the event that $f^k(x_0, BSG(s))$ equals $f^k(x_1, BSG(s))$. Consider the following $(2n, n, v(\mathcal{H}))$-LBP for the input pair (x_0, x_1):

- Input: The LBP starts at a node labeled by (x_0, x_1).

[11] Note that the existence of such a generator follows by Theorem 2.15.

- Layer i (for $i \in [n]$): Get random input $h_i \in \mathcal{H}$ and move from node (f_0^{i-1}, f_1^{i-1}) to the node labeled (f_0^i, f_1^i) with $f_0^i = f(h_i(f_0^{i-1}))$ and $f_1^i = f(h_i(f_1^{i-1}))$.
- The LBP accepts if $f_0^n = f_1^n$ and rejects otherwise.

The LBP described above has parameters $s = 2n$, $t = n$ and $v = 2n$. Furthermore, it accepts with probability that is exactly the desired collision probability, that is, the probability that $f^k((x_0, \overline{h})) = f^k((x_1, \overline{h}))$ over *any* distribution on $\overline{h} = (h_1, \ldots, h_k)$. For every pair (x_0, x_1) with $f(x_0) \neq f(x_1)$ this probability over random $\overline{h}$ was bounded in the proof of Lemma 3.6 by:

$$\Pr_{\overline{h} \leftarrow \mathcal{H}^k}[f^k(x_0, \overline{h}) = f^k(x_1, \overline{h})] \leq \frac{k}{|Im(f)|}$$

Since the generator fools the above LBP, then replacing the random inputs $\overline{h}$ with the output of the bounded-space generator does not change the probability of acceptance by more than $\varepsilon_A(n) = 2^{-n}$. Therefore:

$$\Pr_{s \leftarrow U_{\tilde{n}}}[f^k(x_0, BSG(s)) = f^k(x_1, BSG(s))] \leq \frac{k}{|Im(f)|} + \frac{1}{2^n} \leq \frac{k+1}{|Im(f)|}$$

When taking the probability over random (x_0, x_1) we also add the probability that $f(x_0) = f(x_1)$. Thus

$$\Pr_{x_0, x_1 \leftarrow U_n, s \leftarrow U_{\tilde{n}}}[f^k(x_0, BSG(s)) = f^k(x_1, BSG(s))] \leq \frac{k+1}{|Im(f)|} + \frac{1}{|Im(f)|} = \frac{k+2}{|Im(f)|}$$

When the above is plugged into the calculation of the collision-probability of the function g (recall that $g(x, s) = (f^k(x, BSG(s)), s)$), we get:

$$\mathrm{CP}(g(U_n, U_{\tilde{n}})) \leq \frac{k+2}{2^{\tilde{n}} \cdot |Im(f)|} \tag{3}$$

On the other hand, we check the probability of getting a collision inside the set T. We first request that both $(x_0, \tilde{h_0}) \in T$ and $(x_1, \tilde{h_1}) \in T$, which happens with probability at least γ^2. Then once inside T, we know that the probability of collision is at least $1/|T|$. Altogether:

$$\mathrm{CP}(g(U_n, U_{\tilde{n}})) \geq \gamma^2 \cdot \frac{1}{|T|} \tag{4}$$

Combining (3) and (4) we get

$$\frac{|T|}{2^{\tilde{n}} |Im(f)|} \geq \frac{\gamma^2}{k+2} \ .$$

But the probability of getting a value in T when choosing a random element in $Im(f) \times \{0,1\}^{\tilde{n}}$ is exactly $\frac{|T|}{2^{\tilde{n}} \cdot |Im(f)|}$. Thus, $\Pr[(y, s) \in T] \geq \gamma^2/(k+2)$ as requested. □

REMARK 3.14

It is tempting to think that one should replace Nisan/INW generator in the above proof with the generator of Nisan and Zuckerman [NZ96]. That generator may have seed of size $\mathcal{O}(n)$ (rather than $\mathcal{O}(n \log n)$) when $s = 2n$ as in our case. Unfortunately, with such a short seed, that generator will

incur an error $\varepsilon_A(n) = 2^{-n^{1-\gamma}}$ for some constant γ, which is too high for our proof to work. In order for the proof to go through we need that $\varepsilon_A(n) < poly(n)/|Im(f)|$. Interestingly, this means that we get a linear-input construction when the image size is significantly smaller than 2^n. In order to achieve a linear-input construction in the general case, we need better generators against LBPs (that have both short seed and small error).

3.4.1 Dealing with Non Length-preserving Functions

The pseudorandom generators presented in this section assumed that the underlying regular one-way function is length-preserving. We mention that this is not a necessity and outline how any regular one-way function can be used. For the simple case that f is shrinking, simply padding the output to the same length is sufficient. The more interesting case is of a length-expanding one-way function f. The important point is that we want the generator to be almost linear in the length of the input to f rather than its output. In Lemma 2.6 we show how to transform an expanding one-way function f from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ (for simplicity we write just ℓ) into a length preserving one-way function from $\{0,1\}^{2n}$ to $\{0,1\}^{2n}$. This construction, however, does not maintain the regularity of the one-way function (it maintains only an approximate regularity).

For the regular case we suggest a different solution. Rather than changing the underlying one-way function to be length preserving, we change the randomizing hash functions to be shrinking. That is, given a regular one-way function $f : \{0,1\}^n \mapsto \{0,1\}^\ell$, define the randomized iterate of this function with respect to a family of hash functions from $\{0,1\}^\ell$ to $\{0,1\}^n$. The randomized iterate is now well defined, and moreover, we can show that the collision probability hardly changes. Previously the only way to introduce a new collision was during the application of f (a collision happened with probability $1/Im(f)$ in each iteration). In the new construction a collision can also be introduced when applying a hash function. But such a collision during hashing happens with probability only 2^{-n} (by the pairwise independence of the hash). Thus the overall collision probability at most doubles and the proof of security follows.

The only problem with this approach is that the description of such hash functions is too long (it is $\mathcal{O}(\ell)$ instead of $\mathcal{O}(n)$). This is overcome by using an efficient family of *almost* pairwise-independent hash functions from ℓ bits to n bits with error 2^{-n} (see Definition 2.2), which requires a description of only $\mathcal{O}(n)$ bits.

4 Pseudorandom Generator from an Exponentially Hard One-way Function

4.1 Overview

4.1.1 The randomized iterate and general one-way functions

As mentioned in the introduction, the last randomized iteration of a general one-way function is not necessarily hard to invert and in fact may be easy to invert. However, this hardness is not totally diminished, it simply deteriorates in every additional iteration. By refining the techniques used in the case of regular one-way functions we manage to give a lower bound on this deterioration of the hardness to invert the function. More precisely, we show in Section 4.2 that there exists a set S^k of inputs to f^k with density at least $\frac{1}{k}$ such that the k'th randomized iteration is hard to

invert over inputs taken from S^k. As a result, a hard core bit of the k'th randomized iteration has the guarantee that with probability at least $\frac{1}{k}$ it is pseudorandom.

4.1.2 The multiple randomized iterate

Our goal is to get a string of pseudorandom bits, and the idea is to run m independent copies of the randomized iterate (on m independent inputs). We call this the ***multiple randomized iterate***. From each of the m copies we output a hardcore bit of the k'th iteration. This forms a string of m bits, of which $\frac{m}{k}$ are expected to be random looking. The next step is to run a randomness extractor on such a string (where the output of the extractor is of length, say, $\frac{m}{2k}$). This ensures that with very high probability, the output of the extractor is a pseudorandom string of bits.

4.1.3 The pseudorandom generator - a first attempt

A first attempt for the pseudorandom generator runs the multiple randomized iterate (on m independent inputs) for d iterations. For each $k \in [d]$ we extract $\frac{m}{2k}$ bits at the k'th iteration. These bits are guaranteed to be pseudorandom (even when given all of the values at the $(k+1)$'st iterate and all of the randomizing hash functions). Thus outputting the concatenation of the pseudorandom strings for the different values of k forms a long pseudorandom output (by a standard hybrid argument).

This concatenation, however, is still not long enough. It is required that the output of the generator is longer than its input, which is not the case here. The input contains m strings $x_1, \ldots, x_m$ and md hash functions. The hash functions are included in the output, so the rest of the output needs to make up for the mn bits of $x_1, \ldots, x_m$. At each iteration we output $\frac{m}{2k}$ bits which adds up to $\sum_{k=1}^{d} \frac{m}{2k}$ bits. This is a harmonic progression that is bounded by $m\frac{\log d}{2}$ and in order to exceed the mn lost bits of the input, we need $d > 2^n$ which is far from being efficient.

4.1.4 The pseudorandom generator and exponential hardness

The failed generator from above can be remedied when the exponential hardness comes into play. It is known that if a function is 2^{cn}-one-way (for some constant $c \in (0,1)$), then it has a $2^{c'n}$-hardcore function of $c'n$ bits (for another constant c'). Thus, if the original hardness was exponential, then in the k'th iteration we can actually extract $c'n$ random looking strings, each of length $\frac{m}{2k}$. Altogether we get that the output length is $c'n\sum_{k=1}^{d}\frac{m}{2k} \geq c'mn\log d$. Thus for a choice of d such that $\log d > c'$, we get that the overall output is a pseudorandom string of length greater than the input. The input length of the construction is $\mathcal{O}(nm)$, where m can be taken to be approximately $\mathcal{O}(\log \frac{d}{\varepsilon(n)})$ where $\varepsilon(n)$ is the security of the resulting generator. In particular, in order to get a $2^{\Omega(n)}$-pseudorandom-generator one needs to take a seed of length $\mathcal{O}(n^2)$.

To sum up, we describe the full construction in a slightly different manner: One first creates a matrix of size $m \times d$, where each *row* in the matrix is generated by computing the first d randomized iterates of f (each row takes independent inputs). Now from each entry in the matrix $\mathcal{O}(cn)$ hardcore bits are computed (thus generating a matrix of hardcore bits). The final stage runs a randomness extractor on each of the *columns* of the hardcore bits matrix.[12] Moreover, the number

[12] Note that each execution of the extractor runs on a column in which each entry consists of a single bit (rather than $\mathcal{O}(cn)$ bits). This is a requirement of the proof technique.

of pseudorandom bits extracted from a column deteriorates from one iteration to another ($\frac{m}{k}$ pseudorandom bits are taken at the columns associated with the k'th randomized iterate).

4.1.5 Some Remarks

- Our method also works for $2^{\phi(n)}$-one-way functions as long as $\phi \in \Omega(\frac{n}{\log n})$. Loosely speaking, this is because for large values of d, the value $\frac{1}{d}$ becomes too small to overcome with limited repetition (and thus requires m to grow substantially).
- The description in this section focuses on length-preserving one-way functions, the results can be generalized, using Lemma 2.6, to use non-length preserving functions.

4.2 The Last Randomized Iterate is (sometimes) Hard to Invert

We now formally state and prove the key observation, that there exists a set of inputs of significant weight for which it is hard to invert the k'th randomized iteration even if given access to all of the hash functions leading up to this point.

LEMMA 4.1
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $(t(n), \delta(n))$-one-way, let $k \in \text{poly}(n)$, and let f^k and $\mathcal{H}$ be as in Definition 3.1. Finally, let

$$S^k \stackrel{\text{def}}{=} \left\{ (x, \overline{h}) \in (\{0,1\}^n \times \mathcal{H}^k) \mid D_f(f^k(x,\overline{h}) \geq \max_{j \in [k]} D_f(f^j(x,\overline{h})) \right\} \ .$$

Then there exists a polynomial p, such that for every algorithm A of running-time at most $(t(n) - p(n))$ it holds that

$$\Pr_{(x,\overline{h}) \leftarrow S^k}[A(f^k(x,\overline{h}),\overline{h}) = f^{k-1}(x,\overline{h})] \leq \sqrt[3]{32nk^2\delta(n)} \ ,$$

where the probability is also taken over the random coins of A.

Proof. The proof follows similar lines to the proof of Lemma 3.3. We start by showing that S^k is not too small. By the pairwise independence of the randomizing hash functions $\overline{h} = (h^1, \ldots, h^k)$ we have that for each $0 \leq i \leq k$, the value $f^i(x,\overline{h})$ is independently and randomly chosen from the distribution $f(U_n)$. Thus, simply by a symmetry argument, the k'th (last) iteration has the heaviest preimage size with probability at least $\frac{1}{k}$. Hence,

$$\Pr[(U_n, H^k) \in S^k] \geq \frac{1}{k} \tag{5}$$

Let A be an algorithm with running-time $t_A(n)$ such that

$$\Pr_{(x,\overline{h}) \leftarrow S^k}[A(f^k(x,\overline{h}),\overline{h})) = f^{k-1}(x,\overline{h})] \geq \varepsilon_A(n) \ ,$$

where $\varepsilon_A(n) > \sqrt[3]{32nk^2\delta(n)}$. We show that unless t_A is large, we can use A in order to violate the hardness of f. Consider the procedure M^A for this task.

ALGORITHM 4.2

Algorithm M^A for inverting f.
Input: $y \in Im(f)$.

1. Choose uniformly at random $\overline{h} \in \mathcal{H}^k$.
2. Apply $A(y, \overline{h})$ to get an output x.
3. Output $\overline{h}_k(x)$.

Letting $p(n)$ be the sampling and evaluation time of $\overline{h}$, we have that the running-time of M^A is at most $t(n)$. The rest of the proof of Lemma 4.1 shows that M^A succeeds with probability at least $\varepsilon_A(n)^3/32nk^2 > \delta(n)$ on uniformly chosen $y \in Im(f)$, and we conclude that $t_A(n) > t(n) - p(n)$.

We start by focusing our attention only on those inputs for which A succeeds reasonably well. The following Markov argument implies that the probability of getting an element in the set that A succeeds on is not very small.

CLAIM 4.3

Let $T_A \subseteq Im(f^k) \times \mathcal{H}^k$ be the subset defined as

$$T_A = \left\{ (y, \overline{h}) \in Im(f^k) \times \mathcal{H}^k : \Pr[f(\overline{h}_k(A(y, \overline{h})) = y] > \varepsilon_A(n)/2 \right\} \ ,$$

then

$$\Pr[(f^k(U_n, H^k), H^k) \in T_A \wedge (U_n, H^k) \in S^k] \geq \varepsilon_A(n)/2k \ .$$

Proof. A simple Markov argument (see the proof of Claim 3.5) shows that

$$\Pr_{(x,\overline{h}) \leftarrow S^k}[A(f^k(x, \overline{h}), \overline{h})) = f^{k-1}(x, \overline{h}) \wedge f^k(x, \overline{h}), \overline{h}) \in T_A] \geq \varepsilon_A(n)/2 \ ,$$

and since S^k is not too small, the proof of the claim follows. □

Now that we identified a heavy subset of the inputs that A succeeds upon, we want to say that M^A has a fair chance to hit outputs induced by this subset. This is formally shown in the following lemma.

LEMMA 4.4

For every set $T \subseteq Im(f^k) \times \mathcal{H}^k$, if

$$\Pr[(f^k(U_n, H^k), H^k) \in T \wedge (U_n, H^k) \in S^k] \geq \delta \ ,$$

then

$$\Pr[(f(U_n), H^k) \in T] \geq \frac{\delta^2}{2(k+1)n} \ .$$

Assuming Lemma 4.4 we may conclude the proof of Lemma 4.1. By Lemma 4.4(taking $T = T_A$ and $\gamma = \varepsilon_A(n)/2k$) yields that $\Pr[(f(U_n), H^k) \in T_A] \geq \varepsilon_A(n)^2/16nk^3$. Thus M^A has a $\varepsilon_A(n)^2/16nk^3$ chance of hitting the set T_A on which it will succeed with probability at least $\varepsilon_A(n)/2$. Altogether, M^A succeeds in inverting f with probability $\varepsilon_A(n)^3/32nk^3 > \delta(n)$. □

Proof. (of Lemma 4.4) Divide the outputs of the function f into n slices according to their preimage size. The set T is divided accordingly into n subsets. For every $j \in [n]$ define the j'th slice $T_j = \{(z, \overline{h}) \in T \mid D_f(z) = j\}$. We divide S^k into corresponding slices as well, define the j'th slice as $S^k_j = \{(x, \overline{h}) \in S^k \mid D_f(f^k(x, \overline{h})) = j\}$ (note that since $S^k_j \subseteq S^k$ for each $(x, \overline{h}) \in S^k_j$, for each $0 \leq r < k$ it holds that $D_f(f^r(x, \overline{h})) \leq D_f(f^k(x, \overline{h})) = j$). The proof of Lemma 4.4 follows the methodology of the analogous lemma for the regular case (Lemma 3.6). More precisely, the proof studies the collision-probability of f^k, only here we look at f^k when restricted to S^k_j (that is, we work separately on each slice). Denote this as:

$$\begin{aligned} & \mathrm{CP}(f^k(U_n, \mathcal{H}^k) \wedge S^k_j) \\ = \; & \Pr[(f^k(x_0, \overline{h}_0), \overline{h}_0) = (f^k(x_1, \overline{h}_1), \overline{h}_1) \wedge (x_0, \overline{h}_0), (x_1, \overline{h}_1) \in S^k_j] \;, \end{aligned}$$

where $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$ are uniformly, and independently, chosen in $\{0,1\}^n \times \mathcal{H}^k$. We first give an upper-bound on this collision-probability.

CLAIM 4.5

$$\mathrm{CP}(f^k(U_n, \mathcal{H}^k) \wedge S^k_j) \leq \frac{k+1}{|\mathcal{H}|^k \, 2^{n-j}}$$

Proof. For every two inputs $(x_0, \overline{h}_0)$ and $(x_1, \overline{h}_1)$, in order to have a collision we must first have that $\overline{h}_0 = \overline{h}_1$ which happens with probability $(1/|\mathcal{H}|)^k$. Given that $\overline{h}_0 = \overline{h}_1 = \overline{h}$ (with $\overline{h} \in \mathcal{H}^k$ being uniform), we require also that $f^k(x_0, \overline{h})$ equals $f^k(x_1, \overline{h})$.

If $f(x_0) = f(x_1)$ then a collision is assured. Since it is required that $(x_0, \overline{h}_0) \in S^k_j$ it holds that $D_f(f(x_0)) \leq D_f(f^k(x_0, \overline{h})) = j$ and therefore $\left|f^{-1}(f(x_0))\right| \leq 2^j$. Thus, the probability for that $x_1 \in f^{-1}(f(x_0))$ (and thus of $f(x_0) = f(x_1)$) is at most 2^{j-n}. Otherwise, there must be an $j \in [k]$ for which $f^{j-1}(x_0, \overline{h}) \neq f^{j-1}(x_1, \overline{h})$ but $f^j(x_0, \overline{h}) = f^j(x_1, \overline{h})$. Since $f^{j-1}(x_0, \overline{h}) \neq f^{j-1}(x_1, \overline{h})$, then due to the pairwise-independence of $\mathcal{H}$, the values $\overline{h}_j(f^{j-1}(x_0, \overline{h}))$ and $\overline{h}_j(f^{j-1}(x_1, \overline{h}))$ are uniformly random values in $\{0,1\}^n$, and thus $f(\overline{h}_j(f^{j-1}(x_0, \overline{h}))) = f(\overline{h}_j(f^{j-1}(x_1, \overline{h})))$ also happens with probability at most 2^{j-n}. Altogether,

$$\mathrm{CP}(f^k(U_n, \mathcal{H}^k) \wedge S^k_j) \leq \frac{1}{|\mathcal{H}|^k} \sum_{j=0}^{k} 2^{j-n} \leq \frac{k+1}{|\mathcal{H}|^k \cdot 2^{n+-j}} \;.$$

□

On the other hand, we give a lower-bound for the above collision-probability. We seek the probability of getting a collision inside S^k_j and further restrict our calculation to collisions whose output lies in the set T_j (this further restriction may only reduce the collision probability and thus the

lower bound holds also without the restriction). For each slice, denote $\delta_j = \Pr[(f^k(x,\overline{h}),\overline{h}) \in T_j \wedge (x,\overline{h}) \in S_j^k]$. In order to have this kind of collision, we first request that both inputs are in S_j^k and generate outputs in T_j, which happens with probability δ_j^2. Then once inside T_j we require that both outputs collide, which happens with probability at least $\frac{1}{|T_j|}$. Altogether:

$$\mathrm{CP}(f^k(U_n,\mathcal{H}^k) \wedge S_j^k) \geq \frac{\delta_j^2}{|T_j|} \tag{6}$$

Combining Claim 4.5 and Equation 6 we get:

$$\frac{|T_j| \cdot 2^{j-n-1}}{|\mathcal{H}|^k} \geq \frac{\delta_j^2}{2(k+1)} \tag{7}$$

However, note that when taking a random output z and independent hash functions $\overline{h}$, the probability of hitting an element in T_j is at least $2^{j-n-1}/|\mathcal{H}|^k$ (since each output in T_j has preimage at least 2^{j-1}). This means that $\Pr[(z,h) \in T_j] \geq |T_j| \cdot 2^{j-n-1}/|\mathcal{H}|^k$ and by Equation (7) we deduce that $\Pr[(z,h) \in T_j] \geq \frac{\delta_j^2}{2(k+1)}$. Finally, the probability of hitting T is $\Pr[(z,h) \in T] = \sum_j \Pr[(z,h) \in T_j] \geq \sum_j \delta_j^2 2(k+1)$. Since $\sum_j \delta_j^2 \geq (\sum_j \delta_j)^2/n$ and (by definition) $\sum_j \delta_j = \delta$, it holds that $\Pr[(z,h) \in T] \geq \frac{\delta^2}{2(k+1)n}$ as claimed. □

4.2.1 A Hardcore Function for the Randomized Iterate

A hardcore function of the k'th randomized iteration is taken as a simple generalization of the Goldreich-Levin hardcore function [GL89]. The number of bits taken in this construction depends on the hardness of the function at hand (that is on the hardness of inverting the last iteration of the randomized iterate). Thus, combining Lemma 4.1 regarding the hardness of inverting the last iteration and Theorem 2.9, we get the following lemma.

LEMMA 4.6

Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $t(n)$-one-way function, let $k \in \mathrm{poly}(n)$, and let f^k and S^k be as in Lemma 4.1. Let $\{gl_i\}$ and c be as in Theorem 2.9 and let $\ell_k = \lfloor \frac{c}{4}(\log(t(n)) - \log(k)) \rfloor$.

Then there exist a polynomial p such that the following holds. The function $hc^k : Dom(f^k) \times \{0,1\}^{2n} \mapsto \{0,1\}^\ell$ defined as $hc^k(x,\overline{h},r) = gl_{\ell_k}(f^{k-1}(x,\overline{h}),r)$, is a $t'(n)$-hardcore function of $\widetilde{f^k}(x,\overline{h},r) = (f^k(x,\overline{h}),\overline{h},r)$ over S^k, for any t' satisfying $p(n,t'(n)) < t(n)$.

The following corollary will enable us to apply our extraction lemma (Lemma 2.7) to the hardcore function hc^k.

COROLLARY 4.7

Let f, ℓ_k, hc^k and $\widetilde{f^k}$ be as in Lemma 4.6, then there exist a polynomial p such that the following holds. For any constant $\beta \in (0,1)$ and $i \in [\ell_k]$, the predicate $hc_i^k(z) = hc^k(z)_i$ (i.e., the i'th bit of $hc^k(z)$) is a $(t'(n), 1-\frac{\beta}{k})$ hardcore predicate of the function $\widehat{f^k}(x,\overline{h},r) = (\widetilde{f^k}(x,\overline{h},r), hc^k(x,\overline{h},r)_{1,\ldots,i-1})$, for any t' satisfying $p(n,t'(n)) < t(n)$.

Proof. By the pairwise independence of $\mathcal{H}$, we have that for each $0 \leq j \leq k$ the value $f^j(x, \overline{h})$ is independently and randomly chosen from the distribution $f(U_n)$. Thus, simply by a symmetry argument, the k'th (last) iteration has the heaviest preimage size with probability at least $\frac{1}{k}$. It follows that $\Pr[(U_n, H^k) \in S^k] \geq \frac{1}{k}$, and the proof follows by Lemma 4.6. □

4.3 The Multiple Randomized Iterate

In this section we consider the function $f^{m \times k}$ which consists of m independent copies of the randomized iterate f^k.

CONSTRUCTION 4.8 (the k'th multiple randomized iterate)
Let $m, k \in \mathbb{N}$, and let f^k and $\mathcal{H}$ be as in Definition 3.1. We define the k'th Multiple Randomized Iterate $f^{m \times k} : \{0,1\}^{mn} \times \mathcal{H}^{mk} \mapsto Im(f)^m$ as:

$$f^{m \times k}(\overline{x}, V) = f^k(\overline{x}_1, V_1), \ldots, f^k(\overline{x}_m, V_m),$$

where $\overline{x} \in \{0,1\}^{mn}$ and $V \in \mathcal{H}^{m \times k}$. We let H^{mn} be the random variable uniformly distributed over $\mathcal{H}^{mn}$.

For each of the m outputs of $f^{m \times k}$ we look at its hardcore function hc^k. By Lemma 4.6 it holds that m/k of these m hardcore strings are expected to fall inside the "hard-set" of f^k (and thus are indeed pseudorandom given $(f^{m \times k}(\overline{x}, V), V)$). The next step is to invoke a randomness extractor on a concatenation of one bit from each of the different independent hardcore strings. The output of the extractor is taken to be of length $\lfloor \frac{m}{4k} \rfloor$. The intuition being that with high probability, the concatenation of single bits from the different outputs of hc^k contains at least $\frac{m}{2k}$ pseudoentropy. Thus, the output of the extractor forms a pseudorandom string and might serve as a hardcore function of the multiple randomized iterate $f^{m \times k}$.

LEMMA 4.9 (hardcore function for the multiple randomized iterate)
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $t(n)$-one-way function, let $k \in \text{poly}(n)$, let $f^{m \times k}$ and $hc^k : Dom(f^k) \times \{0,1\}^{2n} \mapsto \{0,1\}^{\ell_k}$ be as in Construction 4.8 and Lemma 4.6 respectively, and let $\text{Ext} : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^{\lfloor \frac{m}{4k} \rfloor}$ be a $(\lfloor \frac{m}{2k} \rfloor, \varepsilon_k)$-strong extractor. We define $hc^{m \times k} : Dom(f^{m \times k}) \times \{0,1\}^{3n} \mapsto \{0,1\}^{\ell_k \cdot \lfloor \frac{m}{4k} \rfloor}$ as $hc^{m \times k}(\overline{x}, V, r, y) = (w_1(\overline{x}, V, r, y), \ldots, w_d(\overline{x}, V, r, y))$, where $\overline{x} \in \{0,1\}^{mn}$, $V \in \mathcal{H}^{m \times k}$, $r \in \{0,1\}^{2n}$ and $y \in \{0,1\}^n$, and for every $i \in [\ell_k]$
$w_i(\overline{x}, V, r, y) \stackrel{\text{def}}{=} \text{Ext}(y, (hc^k(\overline{x}_1, V_1), r)_i, \ldots, (hc^k(\overline{x}_m, V_m), r)_i))$. Finally, we let
$\widetilde{f^{m \times k}}(\overline{x}, V, r, y) = (f^{m \times k}(\overline{x}, V), V, r, y)$

Then, there exists a polynomial p such that the following holds. For every $\gamma(n) > m \cdot 2^{-n/4}$ and $t'(n)$ satisfying $p(t'(n), n, m, k, 1/\gamma(n)) < t(n)$, the function $hc^{m \times k}$ is a $\ell_k(\varepsilon_k + \rho_k + \gamma))$-hardcore function of $\widetilde{f^{m \times k}}$, where ρ_k is the probability is the probability that when taking m independent samples in $Dom(f^k)$, less than $\lfloor \frac{m}{2k} \rfloor$ samples are in a fixed subset of density $2/3k$.

Proof. Let A be an algorithm that given $\widetilde{f^{m \times k}}$, distinguishes between a uniform string and the hardcore function with advantage δ_A. By a standard hybrid argument, there exists an algorithm A' and that runs essentially in the same as A, and for some $j \in [\ell_k]$ distinguishes the value of w_j

from random with advantage δ_A/ℓ_k, given $\widetilde{f^{m\times k}}$ and $\{hc^k(\overline{x}_s, V_s), r)_i : s \in [m], i \in [j-1]\}$. The key observation is that given the hardcore properties of hc^k guaranteed by Corollary 4.7, Lemma 2.7 yields that it is hard to distinguish w_j from random with advantage more than $\varepsilon + \rho_k + \gamma$, for any algorithm of running-time that is not too large. Namely, $p'(t_A(n), n, m, k, 1/\gamma(n)) < t^c(n)$, for some polynomial p' and a constant $c \in (0,1)$. We conclude that for the right choice of p it holds that $p(t_A(n), n, m, k, 1/\gamma(n)) < t(n)$. □

4.4 A Pseudorandom Generator from Exponentially Hard One-way Functions

We are now ready to present our pseudorandom generator. After deriving a hardcore function for the multiple randomized iterate, the generator is similar to the construction from regular one-way function. That is, run randomized iterations and output hardcore bits. The major difference in our construction is that, for starters, it uses hardcore functions rather than hardcore bits. More importantly, the amount of hardcore bits extracted at each iteration is not constant and deteriorates with every additional iteration. The following theorem follows immediately by Lemma 4.9 and a standard BMY like indistinguishability argument (see for example the proof of Theorem 3.11).

THEOREM 4.10 (the pseudorandom generator)
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $t(n)$-one-way function and let $d, m \in \text{poly}(n)$. For any $k \in [d]$ let $\text{Ext}_k : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^{\lfloor \frac{m}{4k} \rfloor}$ be a $(\lfloor \frac{m}{2k} \rfloor, \varepsilon_k)$-strong extractor and let $hc^{m\times k} : Dom(f^{m\times k}) \times \{0,1\}^{3n} \mapsto \{0,1\}^{\ell_k \cdot \lfloor \frac{m}{4k} \rfloor}$ be the hardcore function defined in Lemma 4.9 w.r.t. f and Ext_k. We define G as

$$G(\overline{x}, V, r, y) = (hc^{m1}(\overline{x}, V, r, y) \ldots, hc^{md}(\overline{x}, V, r, y), V, r, y) \ ,$$

where $\overline{x} \in \{0,1\}^{mn}$, $V \in \mathcal{H}^{m\times d}$, $r \in \{0,1\}^{2n}$ and $y \in \{0,1\}^n$.

Then there exists a polynomial p such that the following holds. Assuming that $d \in \text{poly}(n)$ and that G stretches its input, then for every $\gamma(n) > m \cdot 2^{-n/3}$ and $t'(n)$ such that $p(t'(n), n, m, d, 1/\gamma(n)) < t(n)$, G is a $(t'(n), dn \cdot \max_{k\in[d]}\{\varepsilon_k + \rho_k + \gamma\})$-pseudorandom generator, where ρ_k is as in Lemma 4.9.

With the appropriate choice of parameters we get the following pseudorandom generators.

COROLLARY 4.11
Let $\phi(n) \in \Omega(n/\log(n))$ and let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $2^{\phi(n)}$-one-way function. Then there exists a value $d = 2^{\Omega(n/\phi(n))}$ such that the following holds. For any $m \in \text{poly}(n)$ such that $m > 8d$, there exists a choice of $\{\text{Ext}_k\}_{k\in[d]}$ for which the function G of Theorem 4.10, w.r.t. f, d, m and $\{\text{Ext}_k\}$, is a $2^{\Omega(\min\{n, m/d^2\})}$-pseudorandom generator. The input length of G is $\mathcal{O}(mnd)$ and it stretches its input by $\Omega(m\phi(n)/d)$. In particular, for $\phi(n) \in \Omega(n)$ and $m = n$, we get a generator with quadratic seed length and exponential hardness.

Proof. Note that the input length of G is $m\,|x| + |V| + |r| + |y|$ and since it is dominated by the description of V it is in $\mathcal{O}(mdn)$. On the other hand, the output length of G is $m(\sum_{k=1}^{d} \ell_k \lfloor \frac{m}{4k} \rfloor) + |V| + |r| + |y|$. Thus, G stretches its input by $(\sum_{k=1}^{d} \ell_k \lfloor \frac{m}{4k} \rfloor - mn \geq (\lfloor c \cdot \log(t(n)) \rfloor (\sum_{k=1}^{d}(1 - \log(k)) \lfloor \frac{m}{4k} \rfloor) - mn$, for some universal constant $c \in (0,1)$. Since we consider super-polynomial values

for $t(n)$, $d \in \text{poly}(n)$ and $m \geq 8d$, G stretches its input by at least $m(\lfloor \frac{c}{8} \log(t(n)) \rfloor \cdot (\sum_{k=1}^{d} \frac{1}{k}) - n)$. It follows that there exist a universal constant $\gamma > 1$, which depends on the above c, such that G stretches its input by $\Omega(m\phi(n)/d)$, for $d = 2^{\gamma n/\phi(n)}$ and $m > 9d$.

For the security of G, we note that for every $k \in [d]$ it holds that $\rho_k \leq \rho_d$, and by the Chernoff bound it follows that $\rho_k \leq 2^{-\Omega(m/d^2)}$. In addition, for the proper choice of $\{\text{Ext}_k\}$ [13] we have that $\text{Ext}_k \leq \text{Ext}_d \leq 2^{-\Omega(\min\{n,m/d\})}$. Thus, Theorem 4.10 yields that G is a $2^{\Omega(\min\{n,m/d^2\})}$-pseudorandom generator. □

5 Pseudorandom Generator from Any One-way Function

Our implementation of a pseudorandom generator from any one-way function follows the route of [HILL99], but takes a totally different approach in the implementation of its initial step. More precisely, we follow the outline of Holenstein [Hol06a], which gave a new proof to [HILL99] and includes a description and proof of a pseudorandom generator with seed length $\mathcal{O}(n^8)$.[14]

The basic building block of the HILL generator is a ***pseudoentropy pair***.[15] A distribution is said to have *pseudoentropy* at least k if it is computationally-indistinguishable from some distribution that has entropy k. Informally, the a pseudoentropy pair is a pair of a function and predicate on the same input with the following property: The pseudoentropy of the predicate's output when given the output of the function is noticeably larger than the real (conditional) entropy of this bit. In their construction, [HILL99] exploit this gap between real entropy and pseudoentropy to construct a pseudorandom generator. We show that the first randomized iterate of a one-way function together with a standard hardcore predicate forms a pseudoentropy pair with better properties than the [HILL99] one. Hence, plugging our pseudoentropy pair as the first step of the HILL construction, results in a better overall construction. Let us now turn to a more formal discussion. We define the pseudoentropy pair as follows.

DEFINITION 5.1 (pseudoentropy pair (PEP))
Let $g : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ and $b : \{0,1\}^n \mapsto \{0,1\}$ be polynomial-time computable functions. The pair (g, b) is a $(t(n), \delta(n), \gamma(n))$-PEP if

1. $\text{H}(b(U_n) \mid g(U_n)) \leq \delta(n)$,
2. b is a $(t(n), 1 - \delta(n) - \gamma(n))$-hardcore predicate of g.

We write that (g, b) is a $(\delta(n), \gamma(n))$-PEP, it is $(t(n), \delta(n), \gamma(n))$-PEP for every polynomial $t(n)$.

[HILL99] show how to use any one-way function in order to construct a (δ, γ)-PEP, where $\delta \in [0, 1]$ is an *unknown* value and γ is a fraction noticeably smaller than $\frac{1}{2n}$ (i.e., smaller than $\frac{1}{2n} - \frac{1}{p(n)}$ for some polynomial p). They then present a construction of a pseudorandom generator using a $(\delta, \frac{1}{\mathcal{O}(n)})$-PEP where δ is *known*. To overcome this gap, the HILL generator enumerates

[13] For example, the leftover hash lemma ([IL89]) yields that $\text{Ext}_k(x, h) = (h(x), h)$, where $\mathcal{H}$ is a family of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^{\lfloor \frac{m}{4k} \rfloor}$, is a good choice.

[14] In [HILL99], an explicit construction was given only for a pseudorandom generator with seed length $\mathcal{O}(n^{10})$. The existence of the more efficient construction was claimed without being presented or proved.

[15] The notion of pseudoentropy pair is implicitly used in [HILL99], and was formally defined in [HHR06b].

all values for δ (up to an accuracy of $\Omega(\frac{1}{n})$), invokes the generator with every one of these values and eventually combines all generators using an XOR of their outputs. This enumeration costs an additional factor of n to the seed length as well as n^3 times more calls to the underlying one-way function.

In Section 5.1 we prove that the first randomized iterate of a one-way function can be used to construct a $(\frac{1}{2}, \gamma)$-PEP, where γ is a fraction noticeably smaller than $\frac{1}{2n}$.[16] In Section 5.2 we show that by combining our PEP with the second part of [Hol06a] construction, we get a pseudo-random generator that is more efficient and has better security[17] than the original construction of [HILL99]/[Hol06a] (the efficiency improves by a factor of n^3 and the security by a factor of n). For comparison, we present the PEP used by [HILL99] in Appendix A.

5.1 A Pseudoentropy Pair Based on the Randomized Iterate

Recall that for a given function f, we have defined (Definition 3.1) its first randomized iterate as $f^1(x,h) = f(h(f(x)))$. We would like to prove that the first iterate of a one-way function gives rise to a PEP . Indeed, since the randomized iterate maintains some of the hardness of a function (Lemma 4.1), we have that with probability $\frac{1}{2} + \Omega(\frac{1}{n})$ it is hard to compute the value of $f(x)$ given the output $f^1(x,h)$. We want to complement this fact by saying that with probability $\frac{1}{2}$ the value of $f(x)$ can be essentially determined from the output. The latter statement is almost true, except that there typically remains a small amount of uncertainty regarding $f(x)$. To overcome this, we add to the output a small amount of random information about $f(x)$. Specifically, we define the *extended randomized iterate* to include some additional random information about $f(x)$. This information (of $\mathcal{O}(\log(n)$ bits) is small enough to diminish any entropy left in $f(x)$ (in $\frac{1}{2}$ of the inputs), yet is not significant enough to drastically change its pseudoentropy.

5.1.1 The extended randomized iterate of any one-way function

For simplicity we assume that the one-way function is length-preserving, the adaptation to any one-way function is done via Lemma 2.6. We use the following extended version of the first randomized iterate.

DEFINITION 5.2 (the extended randomized iterate)
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$, let $m = \lceil 3\log(n) + 8 \rceil$, and let $\mathcal{H}$ and $\mathcal{H}_E$ be two families of pairwise-independent hash functions from $\{0,1\}^n$ to $\{0,1\}^n$ and $\{0,1\}^n$ to $\{0,1\}^m$ respectively. We define g, the extended randomized iterate of f, as:

$$g(x,h,h_E) \stackrel{\text{def}}{=} (f^1(x,h), h, h_E(f(x)), h_E)$$

where $x \in \{0,1\}^n$, $h \in \mathcal{H}$ and $h_E \in \mathcal{H}_E$. In the following we denote by H and H_E the random variables uniformly distributed over $\mathcal{H}$ and $\mathcal{H}_E$ respectively.

REMARK 5.3
We stress that while the role of $\mathcal{H}_E$ in the above construction and the role of $\mathcal{H}$ in the HILL's PEP

[16] Actually, we prove the more general result that $\gamma \in \Omega(\frac{\log t(n)}{n})$, assuming that f is $t(n)$-one-way.

[17] By "security" we mean that the proof of security of the reduction to the underlying one-way function has better parameters, see Section 2.10 for a more detailed discussion.

are syntactically similar (see Appendix A), their actual role is different. In the above construction the purpose of using $\mathcal{H}_E$ is to reveal a small amount of information (i.e., $\mathcal{O}(log(n))$) about $f(x)$, which in turn *slightly* reduces the uncertainty of $f(x)$ when given $g(x,h,h_E)$. Thus $\mathcal{H}_E$ is used to widen the gap between the real and the pseudoentropy of $f(x)$ given $g(x,h,h_E)$. This extra information is not of major importance and indeed even without using $\mathcal{H}_E$ (i.e. using $g'(x,h,h_E) \stackrel{\text{def}}{=} (f(h(f(x))),h)$), we can still construct a PEP, though parameters will not be as good. On the other hand, in the HILL's PEP there are settings where the hash function reveals a significant amount of information (i.e., $\Omega(n)$) about the input of the function. Thus, removing the hash function altogether guarantees no gap between the real and the pseudoentropy, and in particular the resulting pair is not likely to be a PEP.

The heart of this section are the following two lemmata. In Lemma 5.4 we show that with probability $\frac{1}{2}+\Omega(\frac{\log t(n)}{n})$ it is hard to compute the value of $f(x)$ given a random output $g(x,h,h_E)$. While in Lemma 5.6 we show that the value of $f(x)$ is determined w.h.p. by the value of $g(x,h,h_E)$.

LEMMA 5.4

Let f be a $t(n)$-one-way function and let g be as in Definition 5.2. Then there exists a set $V \subseteq Dom(g)$ of density $\frac{1}{2}+\frac{\log t(n)}{3n}$ and a polynomial p such that the following holds. For every algorithm A of running-time $t'(n)$ satisfying $p(n,t'(n)) < t(n)$, it holds that

$$\Pr_{(x,h,h_E)\leftarrow V\times\mathcal{H}_E}[A(g(x,h,h_E)) = f(x)] < \frac{1}{t'(n)} \ ,$$

where the probability is also taken over the random coins of A.

Proof. Let $V \stackrel{\text{def}}{=} \{(x,h,h_E) \in Dom(g) : \mathrm{Deg}_f(f(x)) \leq \mathrm{Deg}_f(f^1(x,h)+\frac{\log t(n)}{2n}\}$ and let R^0 and R^1 be distributed according to $\left\lceil \frac{2\cdot \mathrm{Deg}_f(f(U_n))}{\log t(n)} \right\rceil$ and $\left\lceil \frac{2\cdot \mathrm{Deg}_f(f^1(U_n,H))}{\log t(n)} \right\rceil$ respectively. Since $f(U_n)$ and $f^1(U_n,H) = f(H(f(U_n)))$ are two independent instances of a random variable distributed over $Im(f)$, it follows that R^0 and R^1 are two independent instances of a random variable distributed over $[\lceil 2n/\log t(n)\rceil]$. By symmetry, $\Pr[R^0 \lneq R^1] = \Pr[R^0 \gneq R^1]$ and since the collision-probability of a random variable is minimal when the distribution is uniform, we have that $\Pr[R^0 = R^1] \geq \frac{1}{\lceil 2n/\log t(n)\rceil}$. Combining the above equations yields that $\Pr[R^0 \leq R^1] \geq \frac{1}{2}+\frac{1}{\lceil 2n/\log t(n)\rceil} \geq \frac{1}{2}+\frac{\log t(n)}{3n}$ and thus

$$\begin{aligned} \frac{|V|}{|\{0,1\}^n\times\mathcal{H}\times\mathcal{H}_E|} &= \Pr[\mathrm{Deg}_f(f(U_n)) \leq \mathrm{Deg}_f(f^1(U_n,H)) + \frac{\log t(n)}{2}] \\ \geq \ \Pr[R^0 \leq R^1] &\geq \frac{1}{2}+\frac{\log t(n)}{3n} \ . \end{aligned}$$

In order to prove the hardness of g over V, let A be an algorithm that runs in time $t_A(n)$ and $\Pr_{(x,h,h_E)\leftarrow V\times\mathcal{H}_E}[A(g(x,h,h_E)) = f(x)] \geq 1/t_A(n)$. Therefore, there exists an efficient algorithm A', with oracle access to A, that computes $f(x)$ with probability at least $\frac{1}{2^m t_A}$, given only $(f^1(x,h),h)$, i.e. A' on input $(f^1(x,h),h)$ chooses a random value z for $(h_E, h_E(f(x)))$ and returns $A(f^1(x,h),h,z)$. The proof Lemma 5.4 follows by the next lemma, which is a straight forward generalization Lemma 4.1.

LEMMA 5.5 **(generalization of Lemma 4.1)**
Let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $(t(n), \delta(n))$-one-way, let $k \in \text{poly}(n)$, and let f^k and $\mathcal{H}$ be as in Definition 3.1. For any value of $\text{gap} \in [n]$ we let

$$S^{k,\text{gap}} \stackrel{\text{def}}{=} \left\{ (x, \overline{h}) \in (\{0,1\}^n \times \mathcal{H}^k) \mid \text{Deg}_f(f^k(x,\overline{h}) + \text{gap} \geq \max_{j \in [k]} \text{Deg}_f(f^j(x,\overline{h})) \right\} \ .$$

Then there exists a polynomial p, such that for every algorithm A of running-time at most $(t(n) - p(n))$ it holds that

$$\Pr_{(x,\overline{h}) \leftarrow S^{k,\text{gap}}}[A(f^k(x,\overline{h}),\overline{h}) = f^{k-1}(x,\overline{h})] \leq 2^{\text{gap}} \cdot \sqrt[3]{32nk^2\delta(n)} \ ,$$

where the probability is also taken over the random coins of A.

Hence, the above lemma (taking $k = 2$ and $\text{gap} = \left\lceil \frac{\log t(n)}{2} \right\rceil$) yields that $p(n, t'(n)) > t(n)$, for the right choice of p. □

LEMMA 5.6
Let f be a $t(n)$-one-way function and let g be as in Definition 5.2. Then there exists a set $L \subseteq Dom(g)$ of density at least $\frac{1}{2} + \frac{1}{4n}$ such that the following holds for every $(x, h, h_E) \in L$.

$$\Pr_{(x',h',h'_E) \leftarrow Dom(g)}[f(x') = f(x) \mid g(x',h',h'_E) = g(x,h,h_E)] > 1 - \frac{1}{16n^2} \ .$$

Proof. Let $Heavy = \left\{(x,h,h_E) \in Dom(g) : \text{Deg}_f(f(x)) \geq \text{Deg}_f(f^1(x,h)\right\}$. By symmetry (as in the proof of Lemma 5.4) it holds that $|Heavy| / |Dom(g)| \geq \frac{1}{2} + \frac{1}{2n}$. The following claim states that we can set L to be most of the elements inside $Heavy$.

CLAIM 5.7

$$\begin{aligned} & \Pr\left[\frac{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E) \bigwedge f(U'_n) = f(U_n)]}{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E)]} \geq 1 - \frac{1}{16n^2} \mid (U_n,H,H_E) \in Heavy\right] \\ \geq & \ 1 - \frac{1}{4n} \ . \end{aligned}$$

Thus, the proof of Lemma 5.6 follows by letting $L = \left\{(x,h,h_E) \in Dom(g) : \frac{\Pr[g(U_n,H,H_E)=g(x,h,h_E) \bigwedge f(U_n)=f(x)]}{\Pr[g(U_n,H,H_E)=g(x,h,h_E)]} \geq 1 - \frac{1}{16n^2}\right\}$. Note that indeed,

$$\begin{aligned} & \Pr[(U_n,H,H_E) \in L] \\ \geq & \ \Pr\left[(U_n,H,H_E) \in Heavy \wedge \frac{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E) \bigwedge f(U'_n) = f(U_n)]}{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E)]} \geq 1 - \frac{1}{16n^2}\right] \\ \geq & \ \frac{1}{2} + \frac{1}{2n} - \frac{1}{4n} = \frac{1}{2} + \frac{1}{4n} \ . \end{aligned}$$

□

Proof. (of Claim 5.7) Let (x,h) be a random element inside $\{0,1\}^n \times \mathcal{H}$. The pairwise independence of $\mathcal{H}$ implies that w.h.p. $\frac{\left|\left\{x' \in \{0,1\}^n : f^1(x',h)=f^1(x,h) \wedge f(x') \neq f(x)\right\}\right|}{|\{0,1\}^n|} \leq 2^{\mathrm{Deg}_f(f^1(x,h))-n}$. When considering also a random h_E, it follows that w.h.p $\frac{|\{x' \in \{0,1\}^n : g(x',h,h_E)=g(x,h,h_E) \wedge f(x') \neq f(x)\}|}{|\{0,1\}^n|} \leq 2^{\mathrm{Deg}_f(f^1(x,h))-n-m}$. On the other hand for every $(x,h,h_E) \in Heavy$ it holds that $\frac{\left|f^{-1}(f(x))\right|}{|\{0,1\}^n|} \geq 2^{\mathrm{Deg}_f(f^1(x,h))-n}$, and we conclude that for a random $(x,h,h_E) \in Heavy$, w.h.p. $\frac{\Pr[g(U_n,H,H_E)=g(x,h,h_E) \bigwedge f(U_n)=f(x)]}{\Pr[g(U_n,H,H_E)=g(x,h,h_E)]} \geq 1 - \frac{1}{16n^2}$.

Let us turn to a more formal discussion. Let $x \in \{0,1\}^n$, $(z_1,z_2) \in Im(f) \times \{0,1\}^m$ and define $S_{x,z_1,z_2} = \{(h,h_E) \in \mathcal{H} \times \mathcal{H}_E : g(x,h,h_E) = (z_1,h,z_2,h_E)\}$. For $x' \notin f^{-1}(f(x))$, the pairwise independence of $\mathcal{H}$ and $\mathcal{H}_E$ implies that $\Pr_{(h,h_E) \leftarrow S_{x,z_1,z_2}}[g(x,h,h_E) = g(x',h,h_E)] \leq 2^{\mathrm{Deg}_f(z_1)-n-m}$. Therefore,

$$\Pr_{(h,h_E) \leftarrow S_{x,z_1,z_2}}[g(U_n,h,h_E) = g(x,h,h_E) \bigwedge f(U_n) \neq f(x)] < 2^{\mathrm{Deg}_f(z_1)-n-m} \tag{8}$$

Let I_{x,h,h_E} be the indicator random-variable that equals 1 if $\Pr[g(U_n,h,h_E) = g(x,h,h_E) \bigwedge f(U_n) \neq f(x)] < 8n \cdot 2^{\mathrm{Deg}_f(f^1(x,h))-n-m}$. By (8) and Markov's inequality we get that, $\Pr_{(h,h_E) \leftarrow S_{x,z_1,z_2}}[I_{x,h,h_E} = 1] \geq 1 - \frac{1}{8n}$. The uniform distribution over $\mathcal{H} \times \mathcal{H}_E$ can be viewed as the distribution induced by choosing a random subset $S_{x,z_1,z_2} \subset \mathcal{H} \times \mathcal{H}_E$ according to its density (i.e., S_{x,z_1,z_2} is chosen with probability $|S_{x,z_1,z_2}| / |\mathcal{H} \times \mathcal{H}_E|$) and then choosing a uniform pair (h,h_E) in S_{x,z_1,z_2}. It follows that $\Pr[I_{x,H,H_E} = 1] \geq 1 - \frac{1}{8n}$ (probability here is taken over uniform choice of H and H_E). Recall that $|Heavy| / |Dom(g)| \geq \frac{1}{2} + \frac{1}{2n}$. Thus, by averaging over all $x \in \{0,1\}^n$ we have that $\Pr[I_{U_n,H,H_E} = 1 \mid (U_n,H,H_E) \in Heavy] \geq 1 - \frac{1}{4n}$ (where probability is taken over the uniform choice of $x \in U_n$ and H, H_E). Since $m = \lceil 3\log(n) + 8 \rceil$, it follows that

$$\Pr\Bigg[\Pr[g(U'_n,H,H_E) = g(U_n,H,H_E) \bigwedge f(U'_n) \neq f(U_n)] < \frac{2^{\mathrm{Deg}_f(f^1(U_n,H))-n}}{32n^2} \tag{9}$$
$$\mid \quad (U_n,H,H_E) \in Heavy \Bigg] \geq 1 - \frac{1}{4n} \ .$$

On the other hand, for any $(x,h,h_E) \in Dom(g)$ it holds that $\Pr[g(U_n,h,h_E) = g(U'_n,h',h'_E)] \geq 2^{\mathrm{Deg}_f(f^1(U_n,H))-n-1}$ and we conclude that

$$\Pr\left[\frac{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E) \bigwedge f(U'_n) \neq f(U_n)]}{\Pr[g(U'_n,H',H'_E) = g(U_n,H,H_E)} \leq \frac{1}{16n^2} \mid (U_n,H,H_E) \in Heavy\right]$$
$$\geq \quad 1 - \frac{1}{4n} \ .$$

□

5.1.2 Constructing the pseudoentropy pair

Recall that the PEP consists of a function and a corresponding predicate. For the function we use the extended randomized iterate, and for the predicate we basically take a Goldreich-Levin hardcore bit. The twist here is that unlike a standard GL predicate, we take the hardcore bit from the intermediate value $f(x)$ rather than from the actual input x. While a hardcore bit taken from x has the required computational hardness, it may also have entropy to it. On the other hand, taking the predicate from $f(x)$ is what gives the desired gap between entropy and pseudoentropy. In the

following formal theorem we use a slightly modified version of g to incorporate the randomness required by the hardcore predicate. The function and predicate are proved to form a $(\frac{1}{2}, \frac{\log t(n)}{4n})$-PEP.

THEOREM 5.8
Let f be a $t(n)$-one-way function, and let $\mathcal{H}$, $\mathcal{H}_E$ and g be as in Definition 5.2. For $r \in \{0,1\}^n$, let $g'(x,h,h_E,r) = (g(x,h,h_E),r)$ and $b(x,h,h_E,r) = \langle f(x), r\rangle_2$. Then there exists a polynomial p such that the pair (g',b) is a $(t'(n), \frac{1}{2}, \frac{\log t(n)}{4n}))$-PEP , for every t' satisfying $p(n,t'(n)) < t(n)$.

Proof. The computational side of the theorem follows directly from Lemma 5.4[2] and Theorem 2.9 (i.e., there exists a polynomial p for which b is a $(t'(n), \frac{1}{2} + \frac{\log t(n)}{4n}))$-hardcore predicate of g' for any t' satisfying $p(n,t'(n)) < t(n)$). It is left to prove that $\mathrm{H}(b'(W_n,U_n) \mid g'(W_n,U_n)) \leq \frac{1}{2}$, where W_n is uniformly distributed over $\{0,1\}^n \times \mathcal{H} \times \mathcal{H}_E$. By Lemma 5.6 there exists a set $L \subseteq Dom(g)$ of density $\frac{1}{2} + \frac{1}{4n}$ such that for every $(x,h,h_E) \in L$ it holds that $\Pr_{(x',h',h'_E) \leftarrow Dom(g)}[f(x') = f(x) \mid g(x',h',h'_E) = g(x,h,h_E)] > 1 - \frac{1}{16n^2}$. Hence,

$$\begin{aligned}
& \mathrm{H}(b(W_n,U_n) \mid g'(W_n,U_n)) = \mathrm{H}(b(W_n,U_n) \mid g(W_n),U_n)) \\
\leq\ & (\frac{1}{2} + \frac{1}{4n}) \cdot \mathrm{H}(\frac{1}{16n^2}, 1 - \frac{1}{16n^2}) + (\frac{1}{2} - \frac{1}{4n}) \\
<\ & (\frac{1}{2} + \frac{1}{4n}) \left(\frac{1}{16n^2}(1 - \frac{1}{16n^2})\right)^{1/2} + (\frac{1}{2} - \frac{1}{4n}) < \frac{1}{8n} + \frac{1}{16n^2} + (\frac{1}{2} - \frac{1}{4n}) < \frac{1}{2} \ ,
\end{aligned}$$

where the second inequality is due to the fact that $\mathrm{H}(q,1-q) \leq 2(q(1-q))^{1/\ln(4)}$ (c.f., [Top01, Theorem 1.2]) and since for small enough q (i.e., $q < 1/100$) it holds that $2(q(1-q))^{1/\ln(4)} < (q(1-q))^{1/2}$. □

5.2 The Pseudorandom Generator

The following is adapted from [Hol06a, Lemma 5].

PROPOSITION 5.9
Let $\gamma(n) > \frac{1}{n}$ be a polynomial computable function and let (g,b) be a $(t(n),\delta(n),\gamma(n))$-PEP. Suppose we are given two non-uniform advices α_n, β_n (for any n) satisfying $\alpha_n \leq \delta(n) \leq \alpha_n + \frac{\gamma(n)}{4}$ and $\beta_n \leq \mathrm{H}(g(U_n)) \leq \beta_n + \frac{\gamma(n)}{4}$. Then there exists a polynomial p such that the following holds for every polynomial computable function $\varepsilon : \mathbb{N} \mapsto [0,1]$. There exists a $\Omega(\min\left\{t'(n), \frac{1}{\varepsilon(n)}\right\})$-pseudorandom generator, with input length is $\mathcal{O}(\frac{n^3 \cdot \log \frac{1}{\varepsilon(n)}}{\gamma(n)^2})$, for any t' satisfying $p(n,t'(n)) < t(n)$.

Combining the above proposition and Theorem 5.8 we get the main result of this section.

THEOREM 5.10
Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $t(n)$-one-way function and assume that $\log(t(n))$ is polynomially approximable up to a constant ratio. Then there exists a polynomial p such that the following holds for

any $\varepsilon : \mathbb{N} \mapsto [0,1]$. There exists an $\Omega(\min\left\{t'(n), \frac{1}{\varepsilon(n)}\right\})$-pseudorandom generator, with input length is $\mathcal{O}(\frac{n^7 \cdot \log \frac{1}{\varepsilon(n)}}{(\log t(n))^3})$, for any t' satisfying $p(n, t'(n)) < t(n)$. In particular, by taking $\varepsilon(n) = \frac{1}{t(n)}$ we get a generator with input length $\mathcal{O}(n^7)$.

Proof. Let $\gamma(n) = \frac{\log t(n)}{3n}$. Theorem 5.8 guarantees the existence of a polynomial p' for which of a pair (g, b) which is $(t(n), \frac{1}{2}, \gamma(n))$-PEP for any t' satisfying $p'(n, t'(n)) < t(n)$ (in the following we assume wlog that $\gamma(n)$ is polynomially computable, the case where we can only approximate $\log(t(n))$ easily follows along the same lines). Since the value of $\delta(n)$ is fixed (i.e., $\frac{1}{2}$), we do not need the advice α_n. Therefore, it is left to take care of the non-uniform advice β_n (note that the value of $\mathrm{H}(g(W_n))$ is not necessarily efficiently approximable). The overcome this problem we used the following pseudorandom generators "combiner". (Since the combiner we are using was previously used by other applications, e.g., [HILL99, Proposition 4.17] and [Hol06a, Theorem 1], we only give a high-level overview of its construction and proof.). For any $\lambda \in [0,1]$, let G_λ be the generator resulting from applying Proposition 5.9 with $\beta_n = \lambda$. Let G'_λ be the result of applying the [GGM86] length-extension method to G_λ such that the output length of G'_λ is $m = \lceil 4n/\gamma(n) \rceil$ times longer than the input length of G_λ. Finally let $G(x_1, \ldots, x_m) = \bigoplus_{i=0}^{m} G'_{\frac{i}{m}}(x_i)$. Clearly G is length expanding, polynomial-time computable and has input length $\mathcal{O}(\frac{n^4 \cdot \log \frac{1}{\varepsilon(n)}}{\gamma(n)^3}) = \mathcal{O}(\frac{n^7 \cdot \log \frac{1}{\varepsilon(n)}}{(\log t(n))^3})$. By a standard hybrid argument, any algorithm B of running-time $t_B(n)$, which distinguishes between the output of G from the uniform distribution with advantage $\frac{1}{t_B(n)}$, implies the following for any $i \in [m]$. There exist an efficient algorithm M_i^B, with oracle access to B, that distinguishes between the output of $G_{\frac{i}{m}}$ from the uniform distribution with advantage $\Omega(\frac{1}{t_B(n) \cdot \mathrm{poly}(n)})$. We conclude that for large enough p, it holds that $p(n, t_B(n)) > t(n)$. □

6 Hardness Amplification of Regular One-way Functions

In this section we present an efficient hardness amplification of any regular weak one-way function.

6.1 Overview

As mentioned in the introduction (Section 1.3), the key to hardness amplification lies in the fact that every δ-weak one-way function has a ***failing-set*** for every efficient algorithm. This is a set of density almost δ that the algorithm fails to invert f upon. Sampling sufficiently many independent inputs to f is bound to hit *every* failing-set and thus fail every algorithm. Indeed, the basic hardness amplification of Yao [Yao82] does exactly this. Since independent sampling requires a long input, we turn to use the randomized iterate, which together with the derandomization method, reduces the input length to $\mathcal{O}(n \log n)$.

6.2 The Basic Construction

For simplicity we assume that the underlying weak one-way function is length-preserving, where the adaptation to any regular one-way function is the same as in Section 3. As a first step, we show that $g(x, \overline{h}) = (f^m(x, \overline{h}), \overline{h})$ is a strong one-way function (for the proper choice of m). The basic intuition is that every iteration of the randomized iterate gives a random element in $Im(f)$

and thus these iterations are bound to hit every significantly large failing-set. The actual proof, however, is more subtle than this.

THEOREM 6.1
Let $\delta(n) > 1/\operatorname{poly}(n)$ and let $f : \{0,1\}^n \mapsto \{0,1\}^n$ be a $(t(n), 1-\delta(n))$-one-way function. Let $m = \left\lceil \frac{4n}{\delta(n)} \right\rceil$, and let $\mathcal{H}$ and f^k be as in Definition 3.1. We define $g : \{0,1\}^n \times \mathcal{H}^m \mapsto Im(f) \times \mathcal{H}^m$ as $g(x, \overline{h}) = (f^m(x, \overline{h}), \overline{h})$. Then there exists a polynomial p such that g is a $t'(n)$-one-way for any t' satisfying $p(n, t'(n)) < t(n)$.

Proof. We start by showing that f has a large failing-set, and then show that any algorithm that inverts g too well, contradicts the existence of such a set.

DEFINITION 6.2 (failing-set)
Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$, let $\varepsilon, \delta : \mathbb{N} \mapsto [0,1]$ and let A be an algorithm trying to invert f. The set $S \subseteq Im(f(\{0,1\}^n)$ is a (ε, δ)-failing-set for A, if $\Pr[f(U_n) \in S] \geq \delta(n)$ and $\Pr[A(y) \in f^{-1}(y)] < \varepsilon(n)$ for every $y \in S$.

LEMMA 6.3
Let $\delta(n) > 2^{-n+1}$ and let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a $(t(n), 1-\delta(n))$-one-way function. Then there exists a polynomial p such that every algorithm of running at most $t'(n)$ satisfying $p(n, t'(n)) < t(n)$, has a $(1/t'(n), \delta(n)/2)$-failing-set.

Proof. Let A be an algorithm of running-time at most $t_A(n)$ trying to invert f and let $S \subseteq Im(f)$ be the set of elements that A inverts with probability less than $1/t_A(n)$ (i.e., for each $y \in S$ it holds that $\Pr[A(y) \in f^{-1}(y)] < 1/t_A(n)$). Assume that $\Pr[f(U_n) \in S] < \delta(n)/2$ and consider the algorithm A' for inverting f. On each input ($y \in Im(f)$) the algorithm invokes A for $nt_A(n)$ independent times (with fresh random coins) and returns a preimage of y if at least one of the instances of A finds such a preimage. The running time of A' is $\mathcal{O}(n \cdot t_A(n)(t_A(n) + t_f(n)))$, where $t_f(n)$ is the evaluation time of f. Also, on the elements outside S, it is guaranteed that A' fails to invert f with probability at most $(1 - \frac{1}{t_A(n)})^{nt_A(n)} < 2^{-n}$. Thus, overall A' succeeds in inverting f with probability at least $1 - \frac{\delta(n)}{2} - 2^{-n} > 1 - \delta(n)$. it follows, by the properties of f, that the running-time of A' has to be at least $t(n)$. Thus, for the right choice of p we get $p(n, t_A(n)) > t(n)$ as required. □

Let A be an algorithm that runs in time $t_A(n)$ and inverts g with probability $1/t_A(n)$. The next lemma shows that there exists an algorithm M^A, that runs in time $t_M(n) \leq p(n, t_A(n))$ for some polynomial p, yet has no $(1/t_M(n), \delta(n)/2)$-failing-set for f. It therefore implies that $t_A(n)$ must be large; namely, that g is one-way.

LEMMA 6.4
Let f and g be as in Theorem 6.1. There exists an oracle aided algorithm $M^{(\cdot)}$ and a polynomial p such that the following holds. Let A be an algorithm that runs in time $t_A(n)$ and inverts g with probability $1/t_A(n)$. If algorithm M^A runs in time $t_{M^A}(n) < p(n, t_A(n))$ then for every set S with

$\Pr[f(U_n) \in S] \geq \frac{\delta}{2}$, it holds that $\Pr[M^A(f(U_n)) = f^{-1}(U_n) \wedge f(U_n) \in S] \geq \frac{1}{t_{M^A}(n)}$.

Before proving Lemma 6.4 we first show how to use it for proving Theorem 6.1. The lemma states that for every set S for which $\Pr[f(U_n) \in S] \geq \frac{\delta}{2}$, there exists $y \in S$ such that $\Pr[M^A(y) = f^{-1}(y)] \geq \frac{1}{t_{M^A}(n)}$. Thus, f does not have a $(1/t_M(n), \delta(n)/2)$-failing-set and by Lemma 6.3 it must be that $p'(n, t_M(n)) > t(n)$ for some polynomial p'. □

Proof. (of Lemma 6.4) For every $i \in [m]$ we define M_i^A as the algorithm that tries to use the i'th iteration in order to invert the function f. Specifically:

ALGORITHM 6.5

Algorithm M_i^A for inverting the last-iteration of f^i.

Input: $(y, h_1 \ldots, h_i) \in Im(f) \times \mathcal{H}^i$.

1. Choose uniformly and independently at random $h_{i+1}, \ldots, h_m \in \mathcal{H}$ and let $w = f^{m-i}(y, h_{i+1}, \ldots, h_m)$.
2. Apply $A(w, h_1, \ldots, h_m)$ to get $(x, h'_1, \ldots, h'_m)$.
3. Return $h_i(f^{i-1}(x, h_1, \ldots, h_{i-1}))$.

The algorithm M^A then tries using each of the possible iterations for inverting f. Formally:

ALGORITHM 6.6

Algorithm M^A for inverting f.

Input: $y \in Im(f)$.

1. Choose a random $h_1, \ldots, h_m \in \mathcal{H}$.
2. For each $i \in [m]$ set $x_i = M_i^A(y, h_1, \ldots, h_i)$.
3. If there exists an i such that $f(x_i) = y$ output x_i, otherwise abort.

For clarity we omit the value n whenever it is clear from the context (i.e., we write δ rather than $\delta(n)$). To prove Lemma 6.4 we need to show that M^A succeeds in inverting f over every large enough set S. This follows by first showing (in Claim 6.7) that there exists at least one iteration $i \in [m]$ that M_i^A succeeds on. Namely, M_i^A succeeds in inverting the i'th randomized iteration of f. This is then combined with the fact that inverting the i'th randomized iterate can be used to invert f (as shown in section 3, Lemma 3.6).

CLAIM 6.7

For any subset $S \subseteq Im(f)$ of density at least $\delta/2$ there exists an index $j \in [m]$ for which $\Pr[f(M_j^A(f^j(U_n, H^j), H^j)) = f^j(U_n, H^j) \wedge f^j(U_n, H^j) \in S] \in \Omega(\frac{1}{t_A^2 \cdot m^2})$.

We defer for now the proof of Claim 6.7 and first use it to prove Lemma 6.4. For ease of notation, define $\alpha_{m,A} = max_{j\in[m]}\{\Pr[f(M_j^A(f^j(U_n,H^j),H^j)) = f^j(U_n,H^j) \wedge f^j(U_n,H^j) \in S]\}$. That is, the claim above states that $\alpha_{m,A} \in \Omega(\frac{1}{t_A^2\cdot m^2})$. Let $S \subseteq Im(f)$ be any subset of density at least $\delta/2$. For $i \in [m]$ let $L_i \stackrel{\text{def}}{=} \left\{(y,\overline{h}) \in S \times \mathcal{H}^i : \Pr[f(M_i^A(y,\overline{h})) = y] \geq \alpha_{m,A}/2\right\}$, where the probability is over the random coins of M_i^A (namely, L_i is the set of elements that M_i^A inverts their last iteration w.h.p.). Note that for every $i \in [m]$ it holds that

$$\begin{aligned}
& \Pr[f(M_i^A(f^i(U_n,H^i),H^i)) = f^i(U_n,H^i) \wedge f^i(U_n,H^i) \in S] \\
= \; & \Pr[f(M_i^A(f^i(U_n,H^i),H^i)) = f^i(U_n,H^i) \wedge f^i(U_n,H^i) \in S \wedge (f^i(U_n,H^i),H^i) \in L_i] \\
+ \; & \Pr[f(M_i^A(f^i(U_n,H^i),H^i)) = f^i(U_n,H^i) \wedge f^i(U_n,H^i) \in S \wedge (f^i(U_n,H^i),H^i) \notin L_i] \\
\leq \; & \Pr[(f^i(U_n,H^i),H^i) \in L_i] + \alpha_{m,A}/2 \ .
\end{aligned}$$

Thus, there exists $j \in [m]$ such that $\Pr[(f^j(U_n,H^j),H^j) \in L_j] \geq \alpha_{m,A}/2$. By Lemma 3.6 (letting $T = L_j$) it follows that $\Pr[(f(U_n),H^j) \in L_j] \in \Omega(\alpha_{m,A}^2/m)$. Since M_j^A inverts the last-iteration of each of the elements in L_j with probability $\alpha_{m,A}/2$, it follows that

$$\begin{aligned}
& \Pr[M^A(f(U_n)) = f^{-1}(U_n) \wedge f(U_n) \in S] \\
\geq \; & \Pr[f(M_j^A(f(U_n),H^j)) = f(U_n) \wedge f(U_n) \in S] \in \Omega(\alpha_{m,A}^3/m) = \Omega(\frac{1}{t_A^6\cdot m^7}) \ .
\end{aligned}$$

Finally, since each invocation of M_i^A involves m invocations of f and a single invocation of A,we have that $t_M(n) < p(n,t_A(n))$ for the right choice of p. □

Proof. (of Claim 6.7) Through the rest of this section we allow ourselves to view g and f^m as functions over $Im(f)\times\mathcal{H}^m$ rather than over $\{0,1\}^n\times\mathcal{H}^m$, where for $y \in Im(f)$ we let $f^m(y,\overline{h}) = f^m(x,\overline{h})$ for some $x \in f^{-1}(y)$.[18] For any $(w,\overline{h}) \in Im(g)$ let $B_{w,\overline{h}} \stackrel{\text{def}}{=} \left\{y \in Im(f) : f^m(y,\overline{h}) = w\right\}$. It readily follows from the proof of Claim 3.7 that for any $y \neq y' \in Im(f)$ it holds that $\Pr[f^m(y,H^m) = f^m(y',H^m)] \leq \frac{m}{|Im(f)|}$. Thus, for any $y \in Im(f)$ it holds that $\mathsf{Ex}[|B_{g(y,H^m)}|] = 1 + \sum_{y'\neq y\in Im(f)} \mathsf{Ex}\left[f^m(y',H^m) = f^m(y,H^m)\right] < m+1$. By Markov's inequality $\Pr[|B_{g(x,\overline{h})}| > 4m\cdot t_A] < \frac{1}{2\cdot t_A}$ and therefore

$$\Pr[A(g(U_n,H^m)) \in g^{-1}(g(U_n,H^m)) \wedge |B_{g(U_n,H^m)}| \leq 4m\cdot t_A] > \frac{1}{2\cdot t_A} \tag{10}$$

Now let $S \subseteq Im(f)$ be a set of density $\delta/2$. By the pairwise independence of $\mathcal{H}$ (actually one-wise independence suffices for this part), we have that for every $y \in Im(f)$, $i \in [m]$ and $\overline{h} \in \mathcal{H}^{i-1}$, it holds that $\Pr[f^i(y,(\overline{h},H)) \in S] \geq \delta/2$. Hence, for any $y \in Im(f)$ it holds that $\Pr[\exists i \in [m] \ : \ f^i(y,H^m) \in S] > 1 - 2^{-2n}$ (recall that $m = \lceil\frac{4n}{\delta}\rceil$). By a union bound,

$$\begin{aligned}
& \Pr[\forall y \in B_{g(U_n,H^m)} \ \exists i \in [m] \ : \ f^i(y,H^m) \in S] \\
> \; & 1 - |B_{g(U_n,H^m)}|\cdot 2^{-2n} \geq 1 - 2^{-n}
\end{aligned} \tag{11}$$

Combining (10) and (11) yields that,

$$\Pr\left[\begin{array}{c} A(g(U_n,H^m)) \in g^{-1}(g(U_n,H^m)) \ \wedge \ |B_{g(U_n,H^m)}| \leq 4m\cdot t_A \\ \wedge\ \forall y \in B_{g(U_n,H^m)} \ \ \exists i \in [m] \ : \ f^i(y,H^m) \in S \end{array}\right] > \frac{1}{2\cdot t_A} - 2^{-n} > \frac{1}{4\cdot t_A} \ ,$$

[18] Note that the above is well defined since for any $x,x' \in f^{-1}(y)$ it holds that $f^m(x,\overline{h}) = f^m(x',\overline{h})$.

where the last inequality follows since wlog $t_A < 2^{n/2}$. Since for any $(x', \overline{h}') \in g^{-1}(z)$ it holds that then $f(x') \in B_z$, we have that

$$\Pr\left[\begin{array}{c} A(g(U_n, H^m)) \in g^{-1}(g(U_n, H^m)) \wedge |B_{g(U_n,H^m)}| \leq 4m \cdot t_A \\ \wedge \exists i \in [m] \ : \ f^i(A(g(U_n, H^m))) \in S \end{array}\right] > \frac{1}{4 \cdot t_A} \ .$$

Thus, by an averaging argument there exists an index $j \in [m]$ such that,

$$\Pr[A(g(U_n, H^m)) \in g^{-1}(g(U_n, H^m)) \wedge |B_{g(U_n,H^m)}| \leq 4m \cdot t_A \wedge f^j(A(g(U_n, H^m))) \in S] > \frac{1}{4 \cdot t_A} \ .$$

For $z \in Im(g)$ let $A(z)_1$ be the first part of $A(z) = (x, \overline{h})$ (i.e., x). The regularity of f yields that

$$\Pr[f(A(g(U_n, H^m))_1) = f(U_n) \wedge f^j(A(g(U_n, H^m))) \in S] > \frac{m}{4 \cdot t_A} \cdot \frac{m}{4 \cdot t_A} = \frac{m^2}{16 \cdot t_A^2} \ ,$$

and we conclude that

$$\begin{array}{rl} & \Pr[f(M_j^A(f^j(U_n, H^j), H^j)) = f^j(U_n, H^j) \wedge f^j(U_n, H^j) \in S] \\ = & Pr[f(A(g(U_n, H^m))_1) = f(U_n) \wedge f^j(A(g(U_n, H^m))) \in S] \\ > & \frac{m^2}{16 \cdot t_A^2} \ . \end{array}$$

□

6.3 An Almost-Linear-Input Construction

In this section we derandomize the randomized iterate used in Section 6.2 to get a (strong) one-way function with input length $\mathcal{O}(n \log n)$. We use the bounded-space generator of either [Nis92] or [INW94] (see Theorem 2.15).

THEOREM 6.8

Let f, m, $\mathcal{H}$ and f^m be as in Theorem 6.1. Let $v(\mathcal{H})$ be the description length of $h \in H$ and let $BSG : \{0,1\}^{\tilde{n} \in \mathcal{O}(n \log n)} \mapsto \{0,1\}^{mv(\mathcal{H})}$ be a bounded-space generator that 2^{-2n}-fools every $(2n, m, v(\mathcal{H}))$-LBP. Define $g' : \{0,1\}^n \times \{0,1\}^{\tilde{n}} \mapsto \{0,1\}^n \times \{0,1\}^{\tilde{n}}$ as $g(x, \tilde{h}) = (f^m(x, BSG(\tilde{h})), \tilde{h})$, where $x \in \{0,1\}^n$ and $\tilde{h} \in \{0,1\}^{\tilde{n}}$.

Then there exists a polynomial p such that g' is a $(t'(n), \frac{1}{t'(n)})$-one-way for any t' satisfying $p(n, t'(n)) < t(n)$.

Proof idea: The proof of the derandomized version follows the proof of Theorem 6.1. In the proof we used the following properties of the family $\mathcal{H}$.

Collision-probability - for any $y \neq y' \in Im(f)$ and $i \in \{0, \ldots, m\}$

$$\Pr[f^i(y, H^i) = f^i(y', H^i)] = \frac{i}{|Im(f)|}$$

Hitting - for all $y \in Im(f)$ and $S \subseteq Im(f)$ of density $\delta/2$

$$\Pr[\exists i \in [m] \ : \ f^m(y, H^m) \in S] > 1 - 2^{-2n}$$

Note that the above two properties can be verified by an $(n, m, v(\mathcal{H}))$-LBP. Since, BSG 2^{-2n}-fools such LBP's, the above properties hold with deviation at most 2^{-2n} w.r.t. to $\overline{h} = BSG(\tilde{n})$. Going through the proof of Theorem 6.1, it is not hard to verify that the proof remains valid also when the above deviations are taking into account. (See the proof of Theorem 3.12 for a more detailed proof on a similar derandomization).

Acknowledgments

We are grateful to Oded Goldreich, Moni Naor, Asaf Nussbaum, Eran Ofek and Ronen Shaltiel for helpful conversations. We also thank Tal Moran and Ariel Gabizon for reading a preliminary version of this paper.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36, 2006.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. pages 112–117, 1982.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[DI99] Giovanni Di Crescenzo and Russell Impagliazzo. Security-preserving hardness-amplification for any regular one-way function. In *31st STOC*, pages 169–178, 1999.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[GIL+90] Oded Goldreich, Russell Impagliazzo, Leonid A. Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 318–326, 1990.

[GKL93] Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM Journal on Computing*, 22(6):1163–1175, 1993.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

[Hås90] Johan Håstad. Pseudo-random generators under uniform assumptions. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 387–394, 1990.

[HHR06a] Iftach Haitner, Danny Harnik, and Omer Reingold. Efficient pseudorandom generators from exponentially hard one-way functions. In *Automata, Languages and Programming, 24th International Colloquium, ICALP*, 2006.

[HHR06b] Iftach Haitner, Danny Harnik, and Omer Reingold. On the power of the randomized iterate. In *Advances in Cryptology – CRYPTO 2006*, 2006.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions in *STOC'89* and *STOC'90*.

[HL92] Amir Herzberg and Michael Luby. Pubic randomness in cryptography. In *Advances in Cryptology – CRYPTO '92*, volume 740, pages 421–432. Springer, 1992.

[Hol05] Thomas Holenstein. Key agreement from weak bit agreement. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2005.

[Hol06a] Thomas Holenstein. Pseudorandom generators from one-way functions: A simple construction for any hardness. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, 2006.

[Hol06b] Thomas Holenstein. Strengthening key agreement using hard-core sets - PhD thesis, 2006.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 230–235, 1989.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 12–24, 1989.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–364, 1994.

[IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version in *CRYPTO'89*.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 1993.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[Phi93] Security preserving hardness amplification using PRGs for bounded space. Preliminary Report, Unpublished, 1993.

[Top01] Flemming Topsøe. Bounds for entropy and divergence for distributions over a two-element set. *Journal of Inequalities in Pure and Applid Mathematics*, 2001.

[WC81] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 1981.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. pages 80–91, 1982.

A HILL's Pseudo-Entropy Pair

In the following we complete the picture of Section 5 by presenting the pseudoentropy pair used in [HILL99].

CONSTRUCTION A.1 (the HILL PEP)
Let $f : \{0,1\}^n \mapsto \{0,1\}^{\ell(n)}$ be a one-way function and let b be any hardcore predicate of f. Let $\mathcal{H}$ be an efficient family of pairwise-independent hash functions from $\{0,1\}^n$ to $\{0,1\}^n$. Let $f_{\mathrm{H}}(x,h,i) = (f(x), h_{i+2\lceil log(n)\rceil}(x), h, i)$ and $b_{\mathrm{H}}(x,h,i) = b(x)$, where $x \in \{0,1\}^n$, $h \in \mathcal{H}$, $i \in [n]$ and $h_k(x)$ stands for the first k bits of $h(x)$.

[HILL99] proves that $(f_{\mathrm{H}}, b_{\mathrm{H}})$ is a $(p + \frac{1}{2n}, \alpha)$-PEP, where $p \stackrel{\text{def}}{=} \Pr_{(x,i)\leftarrow(U_n,[n])}[D_f(f(x)) < i]$ and α is any fraction noticeably smaller than $\frac{1}{n}$. The proof goes by showing that it is hard to predict the hardcore-bit of an input element $(x,h,i) \in Im(f_{\mathrm{H}})$ whenever $i \leq D_f(f(x))$. Roughly speaking, the reason is that in such a case, $(h_{i+2\lceil log(n)\rceil}(x), h, i)$ does not contain any noticeable information about x. Thus, it is essentially as hard to predict $b_{\mathrm{H}}(x,h,i) = b(x)$ given $f_{\mathrm{H}}(x,h,i)$ as it is to predict $b(x)$ given $f(x)$. Since the probability that $i = D_f(x)$ is $\frac{1}{n}$, it follows that for any α noticeably smaller than $\frac{1}{n}$ it holds that b_{H} is a $(p+\alpha)$-hard predicate of f_{H}.

On the other hand, by the pairwise independence of $\mathcal{H}$, whenever $i \geq D_f(x)$ there is almost no entropy (i.e., less then $1/2n$) in $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$. Thus, the entropy of $b_{\mathrm{H}}(x,h,i)$ given $f_{\mathrm{H}}(x,h,i)$ is not more than $p + \frac{1}{2n}$.

REMARK A.2
Note that the above b_{H} is not only $(t(n), 1 - \delta(n) - \alpha(n))$-hardcore of f_{H}, but its hardness comes from the existence of a "hardcore-set" of density $\delta + \alpha(n)$. Where the latter is a subset of the input such that the value of b_{H} is computationally unpredictable over it. This additional property was used by [HILL99] original implementation of pseudorandom generator, but it is not required by the new proof due to [Hol06a]. We note, however, that our PEP presented in Section 5 also has such a hardcore set.